ソフトウェア工学 104－1
（1995．5．31）

識別子と内部コード系に着目した日本語による プログラムの可読性の一評価

平田篤志，早川栄一，並木美太郎，高橋延匡

東京農工大学工学部電子情報工学科

本稿では、日本語プログラミングの有効性を明らかにする実験について述べる。
我々の研究室ではフル２バイトコード系を持ち、制限なく日本語文字を扱うことのできる計算機システムを開発し、その上で日本語プログラミングを実践している。日本語プログラミング環境では、仕様書との対応が良く、保守のしやすいプログラムを記述することが可能になる。この日本語プログラミングについて、その特徴点である識別子の言語と内部コード系に着目し、デバッグ、機能拡張、読解それぞれの観点から有効性を評価する実験を行った。実験を行う環境として、デバッグ、機能拡張についてはマシン上で、また、読解実験については紙上で行うことにより、日本語プログラミングの有効性を示すことができた。

An Evaluation of Readability of C Programs Using Japanese Characters through Identifers and Internal Code Set

Atsushi HIRATA,Eiichi HAYAKAWA,
Mitarou NAMIKI and Nobumasa TAKAHASHI

Tokyo University of Agriculture and Technology

This paper describes experiments investigating the validity of Japanese language programming. We have a full 2-byte code system and have developed an OS and C-compiler that has unrestricted use of Japanese characters. It is on this that we are realizing Japanese programming. In this Japanese language programming environment there is a good correspondence between program and specifications, and thus it is possible to produce programs that are easy to maintain. In relation to Japanese language programming, we have focused on the characteristic of the identifier's language and internal code system, carrying out experiments evaluating its effectiveness from the viewpoints of debugging, extension of features, and readability. The experiments into debugging and feature extension were carried out on-line, and by carrying out those into readability off-line, we were able to verify the effectiveness of Japanese programming.

1．はじめに

プログラミングの対象が地域的文化的に広がると、対象や処理を記述するのに日本語が使えないと不便である。このようなことから、制限なく日本語でプログラミングできる環境が必要となる。

我々の研究室では独自に開発した OS の OS/omicron および C 言語の処理系である CAT 上で上記の日本語プログラミング環境を実現し、この環境上でこれまでに相当数のプログラムを作成してきた。

日本語プログラミング環境の特徴とは次のとおりである。

(1) 計算機の内部文字コード系をフル 2 バイトコード系とする
(2) プログラム中の識別子名の記述に日本語文字の使用を可能にする

プログラムの識別子名の記述に日本語が使えれば、記述しやすさの面で次のような効果が期待できる。

・英語で識別子名を考える必要がなくなるので、プログラミングの際に思考の中断がなされなくなる
・仕様書中のキーワードをそのまま識別子名として記述できるため、仕様書とプログラムとの対応がよくなり保守がしやすくなる

読みやすさについての効果は、過去に紙上での読解実験により、プログラムの識別子に用いる言語として、英語よりも日本語を用いた方が可読性が高いことが確認されている[1,4]。

これに対し本稿では、日本語識別子を用いることの有効性について、実際のプログラミング行動のなかで評価をする必要性から、デバッグと機能拡張の観点から日本語プログラムの有効性を評価する実験をマシン上で行なった。

また、従来の計算機システムにおける 1,2 バイト混在コード系と、OS/omicron のフル 2 バイトコード系との計算機内部の文字コード系の違いに着目し、この違いによるプログラムの読みやすさの違いについての評価も行なった。

2．日本語プログラミング環境

OS/omicron 上の日本語プログラミング環境は、言語 C の処理系である CAT を日本語識別子に対応させている。また、既存の計算機システムが内部コード系として 1,2 バイト混在コード系を採用しているのに対して、OS/omicron ではフル 2 バイトコード系を採用し、それにあわせて CAT の文字型変数のサイズを 2 バイトとしている[2,3]。

両コード系の違いは文字列処理や、文字定数の記述の部分に違いが現れる。

その一つ目は、混在コード系では文字列のバイト数が文字数に比例しないため、文字の内容をチェックしながら文字列処理を行なう必要があることである。

また、混在コード系では日本語文字の文字定数を直接記述することができず、1 バイトごとにコード番号で記述しなければならないという制限が生じる。

例えば日本語文字 1 文字のマッチングを行う処理を両コード系で記述すると図 1 に示すような違いがでる。

```
<<フル2バイトコード系>>
if (文字==’つ’) {
    ・・・
}
<<混在コード系>>
if (文字の上位コード番号==0x82) &&
   (文字の下位コード番号==0xC2) {
    ・・・
}
```

図1・両コード系の1文字マッチング処理の記述の違い

このようなことから、日本語文字を扱う

ことを前提とした文字列処理を記述する場合は、フル 2 バイトコード系の方が有効である。

本稿ではこの内部コード系に着目し、両コード系で記述されたプログラムの読み易さの違いを定量的に評価し、フル 2 バイトコード系のプログラムの方が読み易いことを示そうと考えた。

3．実験の目的と方針

(1) 計算機上での影響を明らかにすること

はじめに述べたように、日英識別子の読み易さの違いを評価する実験は、過去の実験では紙上実験で行ったのに対し、本稿ではデバッグ・機能拡張実験という形で評価実験を行うことを考える。その理由は過去の実験では作業環境を紙上に限定したため、すべての被験者が実際の作業環境で読解できたとはいえないこと、読解実験の形式として、読ませたプログラムの内容を記述させることによって理解の確認を行ったために、プログラム内容は識別子名からおおよその検討がつくのでプログラムの読解につながらなかった可能性があったことからである。

デバッグ効率を評価する試みは、過去に紙上で読解実験が行なわれたが、正しくデバッグできた被験者が少なかったために、統計をとれていない。そこで、実際に試行錯誤や実行確認などができるマシン上の作業環境で、デバッグ・機能拡張実験を行なうことにする。

(2) 内部コード系の違いの可読性への影響を明らかにすること

(1) と同様にマシン上で実験を行なうことを考えると、各被験者に内部コード系の異なる 2 台のマシン上で実験を行なってもらわねばならない。マシンが異なれば、当然マシン間でのツール環境や被験者の作業スタイルの違いが伴うおそれがある。したがって、紙上での読解実験で行なうことにする。

実験用プログラムは両コード系の違いがでやすいように文字列処理を行うプログラム課題を用意し、識別子は両コード系ともに日本語識別子に統一する。

4．実験の概要

本稿では日本語プログラムの有効性を明らかにするために、

(1) デバッグ実験
(2) 機能拡張実験
(3) 読解実験

の実験を行った。各実験方法について (1) を 5 章で、(2) を 6 章で、(3) を 7 章で述べる。

なお、各実験において、用意する実験課題プログラムと被験者の振り分け方に注意した。

4.1. 実験課題プログラム

4.1.1. プログラム内容

評価実験課題として用いるプログラムは、過去の評価実験の考察から、内容としては被験者の知識に依存したものはできる限り用いないことにする。

例えば、過去の実験ではキャッシュに関するプログラム課題を用いた。キャッシュのアルゴリズムは知っていればすぐに内容が検討がつくが、知らないと理解するのに相当の時間を要することになる。

4.1.2. 日英識別子の対応関係

日英識別子の違いの比較実験では、プログラム課題の識別子を日英で記述し分けて実験を行う。その日英の対応関係規則を決める。

実際の場面では、英語識別子では英単語

の短縮形を用いたり、日本語識別子では英語識別子より多くの単語が識別子名に付加されている傾向がある。

そのままの記述形態を実験で用いることはできないので、日英の識別子の対応規則として、日英で単語の短縮形を用いないことにする。

また、識別子に含まれる意味や役割を示す単語量が多いほどそれだけ読解の手助けとなる。したがって、読み易さを計る実験として日英で公平になるように、識別子名が持つ単語量は、対応する日英識別子間で統一することにする。

実際には日本語識別子名をまず考え、それを構成する各単語を英語訳したものを英語版識別子とする。

4.1.3. プログラムの規模

プログラムは、いくつにもモジュール分けされたある程度大きなプログラムで実験を行った方が読み易さを計るのに適している。そのようなプログラムではモジュール名や引数名を頭において各モジュールを読解することになり、識別子名が読解に重要な役割をはたすと考えられるからである。

しかし、実験用プログラムの規模としては、大きなものは不適当である。規模については過去の評価実験での結果から 100 行前後が適当であると考える。この行数は被験者にクロスで振分けて実験を行う二つのプログラムがほぼ同じになるようにする。

4.1.4. プログラム中のコメント

プログラムソース単体での読み易さの評価を行うために、評価実験用プログラムにはいっさいのコメントはつけないようにする。

4.2. 実験の流れ

4.2.1. 実験課題プログラムの振分け方法

実験の流れの概要は次の通りである。

評価実験は日本語識別子と英語識別子、フル 2 バイトコード系と混在コード系のように、プログラム課題を記述し分けて読み易さを比較評価する。そのために被験者を二つにグループ分けし、プログラム課題を二つ以上用意し、図 2 のように被験者にプログラムをクロスで振分けて実験をさせる。その作業全体にかかった時間を評価項目として計測、比較し、作業時間の短い方のプログラムの方が読み易いとする。

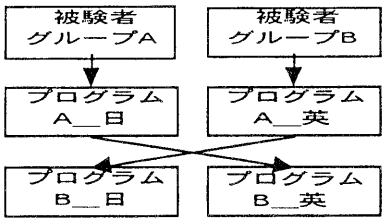

図 2 : 被験者とプログラムの振り分け表

4.2.2. 被験者のグループ分け方法

被験者のグループ分けを行うための基準となる被験者のプログラミング能力として、実験前に被験者本人に次のアンケートをとった。

(1) 現在までに C 言語で作成したプログラムの最大行数
(2) 当研究室内での研究分野

(1) は、記述した量よりも読解した量の方が読解力として適切であるようにみえるが、記述量の方がプログラミング能力を示すパラメータとして適当であると考えた。

被験者は (1) をキーにして行数の多い順にならべて上から二組にグループ分けし、最終的に両グループ間で各研究分野の人間が同人数いるように振分ける。

5．デバッグ実験方法

デバッグ実験は、プログラムにあらかじめバグを混入しておき、そのデバッグを被験者に行わせる。一方、機能拡張実験では与えられたプログラムに拡張すべき機能を追加させる。

できたプログラムの実行確認は、確認に最低限必要な入力データを被験者が考える手間をなくすように、実行確認の際にあらかじめ用意した入力データを自動的に入力して実行確認できるようにしておく。実験はこの実行確認にパスした時点で終了し、それまでにかかった時間を評価項目とする。これは機能拡張実験でも同様である。

実験の評価が細かくできるように、実験の開始終了時間だけを記録するのではなく、エディタ、コンパイラなどのマシン上での各ツールの開始終了時間を記録するようにする。この作業時間を実験中に被験者に記録させるのは困難である。そこで、マシン上のツールが実行されると、そのツールが実行される前後に、現在時刻と実行したツール名が自動的に履歴としてファイルに記録されるようにバッチ処理を組み込んでおく（図 3）。

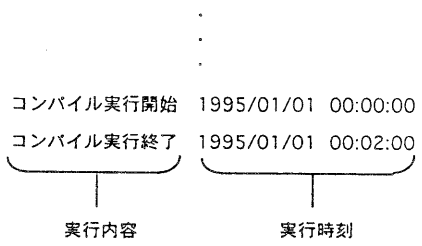

図3・コンパイラを実行したときに
履歴ファイルに追加される内容

デバッグ実験で混入するバグは、被験者がプログラムの読解につながるように、処理のメインとなる部分に混入する。したがって、プログラムの処理内容に関わりの少ない局部的な位置に混入したり、文法的なバグを混入することは望ましくない。そこで、実験で混入するバグとして、ループカウンタの制御部分などに混入する。

バグは 1 箇所だけで、1 行単位の規模のものを混入する。これは、あくまで本来の目的が読解効率を計る実験であり、デバッグにかかる時間の割合を実験全体の時間から少しでも減らすためである。

6．機能拡張実験方法

機能拡張実験では実験課題のプログラムが対応できる入出力データのパターンを増やすタイプの機能拡張を考える。

デバッグ実験と同じく 1 行程度の処理の記述で拡張できる機能を拡張させる。

7．読解実験方法

実験は OS/omicron のフル 2 バイトコード系に対して、混在コード系は MS-DOS のシフト JIS コード系を仮定して実験を行う。

紙上での読解実験は、過去の実験で用いた方法と同じ方法で行なう。

それは、プログラムを被験者に読解させた後に、プログラムの理解を確認するための内容についての設問に回答させ、その作業全体にかかった時間を読解時間として、被験者に計測してもらうという方法である。

設問は次の二つを用意する。

設問 (1)「これは何をするプログラムか」
設問 (2)「次の入力をプログラムに与えたときの出力結果を記せ」

過去の実験では作業時間を、プログラムを読解するのにかかった時間と設問回答にかかった時間を分けて集計したが、設問を見てから読解をはじめる被験者がいたために、分けて集計した意味がなかった。そこで、本稿ではかならず読解してから設問回答させるように被験者に指示し、読解して

設問（1）に回答するまでの時間を読解時間、設問（2）の回答にかかった時間を設問回答時間としてそれぞれ集計することにする。

設問回答形式は、解答群を用意しておくとヒントになってしまうことから、記述回答の形式をとった。

また、混在コード系版プログラムで 2 バイト文字定数の文字がわかるように、実験時に文字コードと文字の対応表を被験者に配ることにする。

8．評価実験

8.1. 被験者とプログラム課題

当研究室内の学部 4 年生と大学院生計 18 人を被験者の対象として、評価実験を試みた。

実験用プログラム課題を四つ用意し、それぞれに次の混入バグ、拡張してもらう機能を次のように用意した。

・デバッグ実験用プログラム課題
①文書中の漢字の割合を出力する
混入バグ：読み飛ばす文字の判定処理部に混入する
②文書中の文の長さの分布を出力する
混入バグ：文の長さを数えるカウンタの値をクリアしない

・機能拡張実験用プログラム課題
③電話番号の数字列にハイフンを挿入する
拡張機能：対応する番号パターンを増やす
④文書中から検索文字列の存在位置を出力する
拡張機能：一致した位置だけでなく一致した文字数も出力する

プログラムはいずれも 60 行前後のものを用意した。

8.2. 実験結果

各プログラムにおける日英版の平均デバッグ・機能拡張時間と、各プログラムについて英語プログラムより日本語プログラムの方が読みやすいことを調べる目的で t 検定を行った結果を表 1 に示す。

表から結論づけると、プログラム③以外のプログラムについて日本語識別子のプログラムの方が読みやすいという結果を得ることができた。

過去の実験結果は紙上読解実験によるものであったが、本稿におけるマシン上実験においても同様に、日本語識別子の方が読みやすいという結果を得ることができた。デバッグ実験をマシン上で行なったことにより、各被験者が本来のスタイルでデバッグ作業を行なうことができたと考えられる。

表1・各プログラムの平均作業時間と検定結果

実験方法	プログラム番号	平均読解時間		t 検定結果
		日本語版	英語版	
デバッグ実験	1	19.4	24.7	自由度 8 で t＝2.02 危険率 4％で日本語の方が読みやすい
	2	14.0	24.3	自由度 8 で t＝5.21 危険率 0.02％で日本語の方が読みやすい
機能拡張実験	3	18.0	14.4	自由度 12 で t＝0.77 危険率 23％で日本語の方が読みやすい
	4	20.1	24.2	自由度 12 で t＝0.79 危険率 23％で日本語の方が読みやすい

（読解時間の単位は分）

8.3. 結果考察

プログラム②は他のプログラムと比較して、はるかに日本語識別子の有効性が高い。これは、他のプログラムと比較して、プログラム内容のキーワードとなる用語の英語訳の意味が分りにくかった、知らなかったせいであると考えられる。

その一方で、マシン上の実験による結果は、識別子の違いによる差だけでなく、被験者のデバッグスタイルの差の要素も含んでしまっていると考えられる。

じっくり読解を行なってからデバッグ、機能拡張を行なうスタイルと、バグ位置や機能拡張位置にあたりをつけてそこを中心に読解を行なうスタイルの差により、いくつかのプログラムで読解時間の分布が二分

してしまっている。

そのことから、プログラム③だけが英語プログラムの方がよいのは、英語版を読解した被験者の大半が後者のスタイルで行なったためであると考えられる。

9．コード系の違いの評価実験

9.1. 被験者とプログラム課題

当研究室内の学部 4 年生と大学院生計 8 人を被験者の対象として、評価実験を試みた。

実験用プログラム課題として文字列処理を行う次の二つを用意した。

⑤漢字熟語の長さの分布を出力する
⑥数え方の表記のゆれを訂正する

プログラムの規模は両プログラムともに 60 行前後のものである。

9.2. 実験結果

各プログラムにおける両コード系の平均読解時間と、各プログラムについて混在コード系版プログラムよりフル 2 バイトコード系版プログラムの方が読みやすいことを調べる目的で t 検定を行った結果を表 2 に示す。プログラムの読解時間は、両コード系のリストの総行数が違うことから、1 行あたりの読解時間で集計した。また、実験ではプログラムの読解時間と設問回答時間を分けて集計したが、有意な結果がみられなかったことから両方の合計時間を読解時間として集計した。

表から結果づけると、両プログラム課題についてフル 2 バイトコード系におけるプログラムの方が読みやすいことを紙上での読解実験により示すことができた。

9.3. 結果考察

プログラム課題⑥は⑤と比較してはるかにフル 2 バイトコード系の可読性が高い。これはプログラム課題⑥ではフル 2 バイトコード系版のプログラム中の文字定数 1 ～ 9、「つ」に着目すれば、プログラムの中身を読解しなくても大体の内容が分るが、混在コード系版のプログラムではこれらの文字定数がコード番号で示されているので、コード番号と文字の対応を調べなければならない。

このことから、プログラム課題⑥では文字定数を直接記述できるかどうかの差がはっきりとでたといえる。

表2・各プログラムにおけるt検定結果

プログラム番号	平均読解時間		t検定結果
	フル2バイトコード系	混在コード系	
5	10．25	21．75	自由度 6でt＝1.50 危険率 9.4％でフル2バイトコード系の方が読みやすい
6	6．5	20．0	自由度 6でt＝2.57 危険率 4.9％でフル2バイトコード系の方が読みやすい

（読解時間の単位は分）

本稿における識別子と内部コード系に着目した評価実験により、日本語プログラミング環境が読み易いプログラムを記述するのに有効であることを示すことができた。

日本語プログラミング環境の実現のためには、はじめに述べたように単にコンパイラを日本語識別子に対応させるだけでなく、計算機システム全体からの日本語対応化をしなければならない。

識別子を日本語に対応させるだけでも英語識別子と比較して可読性が高くなる。このことは前述した過去の読解実験および本稿でのデバッグ・機能拡張実験の定量的な評価により明らかにすることができた。

しかし、それだけでは完全な日本語化として不十分である。計算機システムで日本語文字を扱うための内部文字コード系をフル 2 バイトコード系としなければ、記述時に日本語文字を対象とした文字列処理の記述が複雑化したり、プログラム中での日

本語の使用が制限されることになる。読み易さについても本稿の読解実験により、フル 2 バイトコード系などの固定長バイトコード系の方が有効であるという結果を得られた。

今後の課題は、デバッグ実験で読みやすさを測る場合に、被験者間のデバッグ能力差やスタイルの差を排除した上で実験を行なう必要があることである。

また、本稿の実験のように、当研究室内の日本語プログラミングに慣れた者を被験者の対象とするのではなく、研究室外部の人間を被験者として実験を行う必要がある。

量的な問題としては、被験者数をさらに増やして実験する必要もある。

さらに規模の大きなプログラムで実験を行うことも必要である。それは本稿のように実験時間を限定した実験ではなく、長期的な目でみて例えばデバッグ実験を本稿で使ったツールを利用して行なえば、コンパイル回数や作業時間などを記録し、評価することができると考えられる。

１０．おわりに

本稿における評価実験において、プログラムの識別子に用いる言語として、英語よりも日本語を用いた方がプログラムの可読性が高いことをデバッグ実験により示すことができた。また、文字列処理を行なうプログラムにおいては、フル 2 バイトコード系の方が可読性が高いことを読解実験により示すことができた。

参考文献

[1] 中川正樹，他：“母語プログラミングの理念、実現、実践とその効果”，電子情報通信処理学会論文誌，
Vol.J77-D-I, No.5 , pp.364-374

[2] 並木美太郎，他：“言語 C コンパイラのマルチバイト化の実現方式”，情報処理学会論文誌，
Vol.33, No.11 , pp.1331-1340

[3] 鈴木茂夫，他：“OS/omicron における日本語プログラム環境”，情報処理学会論文誌，
Vol.33, No.11 , pp.1331-1340

[4] 中川正樹，他：“日本語プログラムの可読性の評価と検討”，情報処理学会ヒューマンインタフェース研究会報告，
No.43-1

[5] 佐藤匡正 ：“プログラム変数の命名法”，情報処理学会ソフトウェア工学研究会報告，No.58-15

[6] 落水浩一郎，他：“ソフトウェア開発と保守作業の形態の同質性について”，情報処理学会ソフトウェア工学研究会報告，No.34-5

[7] 四野見秀明，他：“構造化分析／設計の方法論に基づいたプログラム理解支援ツール”，情報処理学会ソフトウェア工学研究会報告，No.87-1

[8] 後藤浩一，他：“ソフトウェアの開発・保守作業の効率改善のためのモジュール分析の試み”，情報処理学会ソフトウェア研究会報告，No.34-19