記　号　処　理 62－1
（1991. 11. 29）

# プログラムを含む高品質ドキュメント作成システム LispWEB

神林 隆
NTT ソフトウェア研究所

*literate programming* の概念を Lisp に適用し、Lisp と TEX を融合させた LispWEB を開発したので報告する。LispWEB システムは、ドキュメントを含む Lisp プログラムをできるだけ読みやすいように出力するものであり、その特徴は、以下の通りである。

- 従来の WEB システムと同様に、ドキュメントとプログラムの混在が可能である。
- TANGLE を省略することにより、LispWEB ソースをそのまま Lisp に読み込ませることができる。
- システム組み込み関数名，プログラマ定義関数名，変数名，定数を出力するフォントを区別している。
- 縦線を使って、括弧の対応を明示する。
- モジュール番号により、関数名を参照できる。

本稿では、LispWEB システムに実現したこれらの機能について述べる。

# LispWEB : Highly readable document preparation system for Lisp

KAMBAYASHI Takashi
NTT Software Laboratories

A concept of *literate programming* is applied to Lisp and a system to prepare highly readable document for Lisp is developed. LispWEB is a combination of Lisp and TEX, and it prepares Lisp programs with documents for printing so that they are as easy to read as possible. Its features include the following:

- Similarly to the usual WEB, we can put documents and programs together in a same file.
- By means of omitting TANGLE, we can get LispWEB source read in Lisp environment directly.
- Via fonts, primitives, programmer defined functions, variables and constants are easily distinguishable.
- Vertical bars are printed to denote matching parentheses.
- We can refer to function names using module numbers.

In this paper, implementations of these facilities in LispWEB system are discussed.

## 1 はじめに

人工知能や記号処理等の分野でよく用いられる言語に、Lispがある。これは、Lispがインタプリタ・ベースで動く言語であり、プロトタイピングを行なうのに適しており、試行錯誤的なプログラミングが容易にできるからである。

しかしその反面、Lispにおけるプログラミングは、作っては捨てての繰り返しになることが多く、それを人に理解してもらおうとするドキュメントが皆無か、また、あったとしてもごくわずかである。事実、コンパイラ・ベースのCやPascalなどに比べると、インタプリタ・ベースのLispに残されているドキュメントは多くはない。表1に、MacLispソース, kclソース, Cコンパイラ, Pascalコンパイラについて、コメントを含む割合を、行数とバイト数で比較した結果を示す。

1984年にGuy L. Steel Jr.たちの手によりCommon Lisp[10]の仕様がまとめられ、以後、それに基づく処理系が数多く作られ、事実上の業界標準となった。Lispは、その汎用性の高さからも、ますます広く使われるようになり、Lispによる実用的規模のシステム開発も盛んに行なわれている。

開発規模が大きくなれば、プログラミングをしている当のプログラマを始めとして、共同作業者や保守担当者など、プログラムを読む人間は予想以上に多くなる。自分で作ったドキュメントつきのプログラムでさえ、しばらくすると分かりにくくなることが多いのに、他人の作ったドキュメントのないプログラムを判読・理解することはかなり労力を要する仕事になる。

そのような状況に陥いることのないように、自分も含めた他の人にも理解しやすいようにLispプログラムを読ませることは、これからの重要な課題となる。そこで、*literate programming*の概念をLispに適用し、プログラミング言語LispとドキュメントTEX用言語TEXを融合させたLispWEBシステムを開発した。本稿では、TANGLEの処理を省略する手法を提案するとともに、プログラム理解を支援する、読みやすいドキュメントの作成方法について報告する。

## 2 literate programming

Stanford大学のDonald E. Knuth教授は「文学作品を書くような意識でプログラムを書くべきである」という主張をしており、これは文芸的プログラミング(*literate programming*[2])と呼ばれている。通常のプログラム作成においては、プログラマは計算機に対して何をすべきかを命じることしか考えていない。計算機が主であり、計算機がプログラムの処理さえできれば、それでよいのである。そのため、プログラム内のコメントは作成者のメモ程度でしかなく、仕様書や設計書はソースプログラムとは別個に管理され、プログラムとの関連性は薄い。

これに対して*literate programming*では、プログラムを自分を含む人間に読んでもらうことを始めから意図しており、プログラマが計算機に対して何をさせたがっているのかを人間に対して十分に説明できなければいけない。そのために、プ

表1: Lisp, C, Pascalのコメントの割合

| | MacLispソース | | kclソース | | Cコンパイラ | | Pascalコンパイラ | |
|---|---|---|---|---|---|---|---|---|
| | 行 | バイト | 行 | バイト | 行 | バイト | 行 | バイト |
| 規模 | 37198 | 1257381 | 20930 | 787737 | 24182 | 746814 | 25747 | 724516 |
| 割合 | 8.7% | 13.9% | 4.9% | 5.4% | 42.5% | 16.1% | 49.0% | 21.2% |

行の割合 = (コメントを含む行数 ÷ 全体の行数) × 100

バイトの割合 = (コメントのバイト数 ÷ 全体のバイト数) × 100

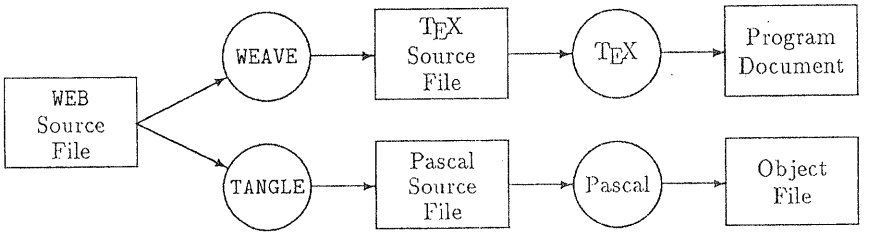

図 1: WEB におけるデータの流れ

ログラムとそれを解説するドキュメントを一体化させ、人間が読みやすいプログラムを作ろうとするものである。 ドキュメント部分には、読者がプログラムを素直に理解できるように、プログラマの意図(計算機に何をさせたいのか)の他に、動作内容、プログラム理解に必要な情報、他のプログラムとの関連、実行結果などを心がけて書くことになっている。

このために、Knuth は TEX[3] 開発プロジェクトの一環として、1981 年にプログラム設計支援システム WEB を作り、*literate programming* の概念を具体化した。 WEB は、プログラミング言語 Pascal と文書処理システム TEX のコードを混ぜ合わせて書けるようになっており、人間と計算機とのコミュニケーションにはプログラミング言語を、人間と人間との情報のコミュニケーションには文書整形用言語を用いて出力したものを使えばよい。

WEB のソースは、プログラムとドキュメントの両方の性質を持っており、同じ WEB ソースから 2 種類の出力 — プログラムコードとドキュメント — を得ることができる。 そのために、図 1のように、WEB には TANGLE と WEAVE という 2 つのプログラムが用意されており、どちらも WEB 自身で記述されている。 TANGLE は計算機のために WEB のソースから Pascal のコードを抽出するものであり、WEAVE は人間にわかりやすいように TEX のソースを抽出するものである。

オリジナルの Pascal 用 WEB の他に、類似システムとして、C プログラマ用の CWEB[6] や Modula-2 用の MWEB[7] などが作られている。 また、言語仕様を与えると、その言語用の WEB システムを生成する SPIDER[8] というプログラムも存在する。

Lisp にも、この *literate programming* の考え方は応用できるはずである。

## 3 LispWEB

LispWEB システムは、コメント(ドキュメント)を含む Lisp プログラムをできるだけ人が読みやすいように出力しようとするものである。 ソースファイルに、Lisp プログラムとその機能等を説明するドキュメント[1]をコメントにして書いておくと、LispWEB システム(図 2での WEAVE)がタイプセットのための処理を行ない、TEX ファイルを作成する。 あとはその TEX ファイルを TEX の処理系にかけてやることにより dvi ファイルができ、読みやすい出力が得られる(図 2参照)。

実際に動くコードとそれを解説するドキュメントが同じファイル上の、しかも同じ位置にあるので、プログラムを理解するのが容易である。 また、文書整形言語として TEX を使用しているので、目次や index などの TEX の持つ様々な機能を駆使して、読みやすいドキュメントを作成することが可能である。

### 3.1 TANGLE の省略

TANGLE は、WEB ソースからプログラム部分だけを取り出し、言語処理系(Pascal, C であればコンパイラ、Lisp であればインタプリタ)が理解できる形にプログラム部分を組み直すプログラム

[1]実際には、LispWEB システムが扱うドキュメントは LATEX[5] の文章になっている必要がある。

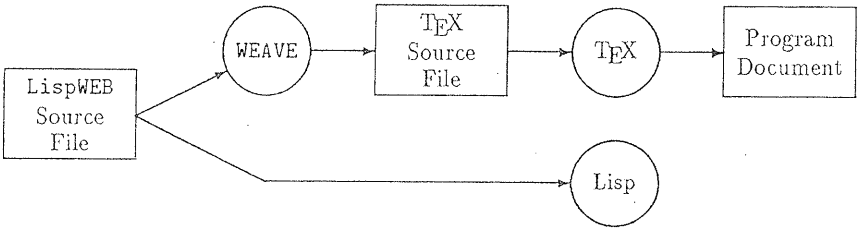


図 2: LispWEB におけるデータの流れ

であり、TANGLE により得られたものが言語処理系によって解釈され、プログラムが動く仕組みになっている。

Lisp においては、プログラムを修正するたびに TANGLE にかけていたのでは、Lisp の利点であるプロトタイピングに支障をきたすことになるので、turn around time を少なくするために、WEB ソースではなく、TANGLE により得られた Lisp プログラムソースを直接編集してデバグすることが多くなると予想される。このような WEB ソースと Lisp プログラムソースの二重化は、プログラミングやデバグ時に混乱をきたす元凶となる。

この混乱を防ぐために提案するのが、TANGLE を省略して、LispWEB ソースと Lisp プログラムソースを同一化する手法である。そのために、プログラマに、ドキュメント部分にセミコロン(";")をつける[2]ことを強いる。強いるとは書いたが、通常の Lisp プログラミングにおいても、セミコロンはコメントを書くために用いられているので、何ら問題はないと思われる。

この手法を用いると、ドキュメント部分はコメントアウトされているので、LispWEB ソースをそのまま Lisp に読み込ませることが可能となる。これは、interactive にプログラミングを行なう Lisp には非常に好都合である。また、TANGLE ファイル(TANGLE により得られるプログラムソース)を経由しないので、従来の WEB で問題になっていたデバグの対象ファイルを、LispWEB のソースファイルそのものに限定することが可能になるという利点もある。

### 3.2 モジュールの取り扱い

モジュールとは、WEB におけるプログラミングの単位であり、いくつかのモジュールが集まって、LispWEB ソースが構成される。個々のモジュールは、そのモジュールの機能を説明するドキュメントと、その機能をインプリメントするプログラムコードから構成される[3]。

従来の WEB では大きな手続きや関数を複数のモジュールに分割してプログラミングを行なっていた。Pascal は 1 パスでコンパイルされるために、「関数名や変数名は使用する前に必ず定義しておかなければならない」という言語仕様上の制約がある。WEB は、この制約を TANGLE を用いて克服した。つまり、プログラマが読みやすいと思う順序でプログラミングしておけば、あとは勝手に TANGLE がバラバラになっているモジュールを並び換えて、ちゃんと動く Pascal のソースを組み上げる仕組みになっている。だから、手続きや関数を分割してプログラミングすることに意義があった。

しかし、このように 1 つの手続きや関数を複数のモジュールに分割してしまうと、局所変数のスコープが紛らわしくなるという問題点がある。ある関数定義内で使われるはずの局所変数が、全く別の位置にある(分割された)モジュールで使用されることになる。そのモジュールでは、その変数

[2] SchemeTEX[9] がこの手法を用いて、ドキュメント部とプログラム部を分けているが、TANGLE プログラムは存在する。SchemeTEX の TANGLE は、コメント行をスキップして、プログラムのみのファイルを作るだけである。

[3] ドキュメントのみ、プログラムのみのモジュールも許される。

が大域変数なのか、局所変数なのか、また局所変数であるとしてもどこで指定されている局所変数なのか、そのスコープを知る由もない。これは、プログラムを理解する時の障壁となりうる。

Lispにおいては、わざわざモジュール分割という方法をとらなくても、関数呼び出しという形で、大きな関数を細分化することが可能である。しかも、プログラムはすべて位置独立[4]に書けるので、Pascalのように定義順序を気にする必要もない。そこで、LispWEBでは基本的に、1関数定義を1モジュールとして扱う[5]ことにし、大きな関数をいくつかに分割したいときは、分割したい部分を新たな関数として定義して(これも1モジュールとなる)、その関数を呼び出す形式をとることにする。これだと、局所変数のスコープが分割されるという心配もなくなる。

## 4 読みやすくするための機能

*literate programming*で本来意図する読みやすさとは、なぜこのプログラムが作られたのか、このプログラムがどのような動きをするのかなどが分かりやすく説明されていることを意味している。しかし、プログラムの見た目の読みやすさも、プログラムを理解する上で重要なことである。読みやすいプログラムを出力するために、以下の点について検討し、LispWEB上にインプリメントを行なった(出力例は付録1、ソース例は付録2を参照のこと)。

### 4.1 フォント

今までLispプログラムの書式は、symbolicsなどのビットマップを使用したLispマシンを除くと、タイプライタ体のみを使って出力するのが通例であった。しかし、タイプレスのLispといえども、データの種類に応じてフォントを区別して出力すれば、読みやすさは向上するはずである。

そこで、区別するべきデータの種類を考察してみた。Lispのプログラムは S式の集まりであり、S式は

$$(func\ arg_1\ arg_2\ \ldots\ arg_n)$$

の形になっている。ここで、S式の中には、関数と引数の区別があることがわかる。さらに関数には、1. **システム組み込みの関数**と2. **プログラマが定義した関数**の2種類が存在し、引数は、3. 変数と4. **定数**とS式の3つの場合に分けられる。S式の場合は、さらに再帰的な構造になる。

この4種類のデータにそれぞれ異なるフォントを割り当てて出力を行なうことを試みた。幸い、TEXには様々なフォントが用意されているので、それらを使うことが可能である。プログラマ定義関数名はボールド体で、システム組み込みの関数名はローマン体で、変数はイタリック体で、stringやquoteされたデータなどの定数はタイプライタ体で出力するように割り当てた。

この結果、

- どの関数がプログラマ定義関数で、どの関数がシステム関数なのか一目瞭然となる。
- 定数データが他のプログラムと区別できる。

などの評価が得られた。

### 4.2 括弧対応の表示

括弧の対応が正しくとれないと、プログラムの動作を正しく理解することはできない。Lispプログラムを読んでいく上で、括弧の対応を取るということは、非常に重要なことである。では、どのようにすれば、プログラムを読む人間にとって括弧の対応が取りやすくなるのか考えてみた。

まず手始めに、

- 括弧の右下に入れ子の深さの番号を振る。

例: ($_0$cond ($_1$($_2$and ($_3$numberp ($_4$cadr *pages*)$_4$)$_3$
($_3$= ($_4$cadr *pages*)$_4$ *num*)$_3$)$_2$ ...

- 太さの違う括弧をいくつか用意し、入れ子が深くなるにつれて細い括弧を使うようにする。

例: (cond ((and (numberp (cadr *pages*))
(= (cadr *pages*) *num*)) ...

[4]関数をどのような順番で定義しても構わないこと。

[5]さらに特別の場合として、トップレベルの関数呼び出しなどで似たような動作をするものは、複数個まとめて1モジュールにできるようになっている。

などの方法を試してみたが、効果はいま一つであった。大きなブロックを読んでいる時に、対応する括弧を瞬時に見つけることができないのである。

そこで着眼点を変えて、括弧を直接扱うのではなく、対応する括弧までのブロックを明示的に表示する方法をとってみた。インデントしているところに縦線を入れることにより、括弧の対応しているブロックを表示することが可能である。このようにすると、縦線を追っていくだけで、対応する括弧を見つけることができるので、プログラムの構造を容易に理解できる。

縦線の効果を調べるために、アイ・カメラによる実験を行なった。Lispプログラミングの経験者に、10kステップ程度のLispプログラムをLispWEBに通したものを読んでもらい、被験者の視線がどのように動くかを観察した。この実験により、被験者がプログラム中の大きなブロックを読んでいる時に、この縦線を目で追ってブロックの始まりと終りを認識していることが確認でき、縦線がないプログラムを読む時に比べて、視点の移動が少なくなるという結果を得た。

また、縦線の間隔が等間隔ではなく不規則になっているので、それが逆に、ページを隔てた時のインデントのパターンマッチングをわかりやすくしているという意見もあった。

### 4.3 関数の定義モジュール番号の表示

Lispプログラムを読む時には、そのプログラム(関数)内で使われている関数をさらに追いかけて調べていくことがほとんどである。その時に、その関数がどこで定義されているかが容易にわかると、検索が楽になるので、プログラムの理解に大いに役立つ。

LispWEBでは、各モジュールにユニークな番号をつけて、モジュール番号による識別を可能としている。このモジュール番号によるcross referenceの機能について考察した。

使用している(呼び出す側の)関数名の右下に、その関数が定義されている(呼び出される側の)モジュールの番号を付けることにより、関数定義の参照を行ないやすくすることが可能である。この機能は、3.2で述べた、1関数を1モジュールとして扱う規則により実現できた。また、"TEX:the program"[4]で用いられているmini index[6]では、視線が一旦ページの右下に移るのに比べて、このcross referenceは、見ている関数から視線を離す必要がないことからも、効果的であると言える。

また、各モジュールの最後には、そのモジュールで定義している関数が使用されているモジュールの番号を表示した。これにより、関数の相互呼び出し関係が、関数の定義モジュールからもわかるようになっている。

### 4.4 Index

前節(4.3)で説明したcross referenceの機能は、グローバルに定義されている関数名と変数名についてのみ動作する。しかし、プログラム内には、局所的にのみ使われる変数も存在する。それら局所変数名も含めたidentifierに関するindexを巻末に付けることにした。さらに、そのidentifierがグローバルに定義されている時は、他と区別できるように、定義されているモジュール番号をボールド体にして先頭に表示するようになっている。

巻末のindex内でも、4.1で述べたフォントの割り当てにしたがい、データの種類に応じてidentifierを表示するフォントは変えられている。したがって、同じ名前のidentiferであっても、それがグローバル関数の時と局所変数の時とで別々にindexされる。

### 4.5 インデント

プログラムを理解する上で、プログラムの構造に適合したインデントを施してやり、プログラムを読みやすくすることは、常套手段である。

[6] 見開きページの右下に、その両ページで使われている関数，マクロ，グローバル変数について、それらが定義されているモジュールの番号や値を表示するindex。

LispWEB システムでは、Emacs と同様なインデントの正規形で、出力を行なう。これは、Emacs が広く使われており、Emacs を用いてプログラミングする人間が多いからである。Emacs プログラマにとっては、編集しているプログラムのインデントと LispWEB を用いて出力したプログラムのインデントの違和感が少ないので、プログラミングしている時と同じ雰囲気で、出力結果を見ることが可能である。

### 4.6 評価と展望

LispWEB システムを、NTT 内で作成・使用中の“DDX 高機能プロトコルアナライザ”(規模は約 8.2k ステップ) のプログラムに適用した。その出力結果は、「ユーザ定義関数とシステム関数の区別が容易である」とか「括弧の対応がつけやすい」などの好意的な意見が得られ、総合的に判断して、実用に十分耐え得るものであるという評価を得た。

今後は、データタイプに応じたタイプセット機能や、interactive な検索を可能にする機能などを付加し、さらには、ドキュメントの作成支援システムとして発展させるつもりである。

LispWEB は kcl を用いて作られており、プログラム規模は約 1.7k ステップである。また、ソースとなるプログラムは、Common Lisp を対象にしている。

## 5 おわりに

本稿では、我々が開発した LispWEB システムについて報告した。まず、*literate programming* の概念について簡単に説明し、interactive なプログラミングが可能である Lisp の特性を活かすために、TANGLE を省略する手法を提案した。さらに、Lisp プログラムを含むドキュメントを読みやすくする機能として、

- データの種類によるフォントの区別
- 縦線による括弧対応の表示
- 関数の定義モジュール番号の表示
- Index
- インデント

について考察し、評価を行なった。

なお、LispWEB は TEX や WEB というフリーソフトウェアを基にして作られたシステムであるので、それ自身もフリーソフトウェアとして希望者には提供することができる。

## 参考文献


[1] Wayne Sewell: “Weaving a Program — Literate Programming in WEB”, Van Nostrand Reinhold, 1989.

[2] Donald E. Knuth: “Literate Programming”, *The Computer Journal*, Vol.27, No.2, pp.97–111, 1984.

[3] Donald E. Knuth: “The TEXbook”, Addison-Wesley, 1986.

[4] Donald E. Knuth: “TEX: The Program”, Addison-Wesley, 1986.

[5] Leslie Lamport: “A Document Preparation System LATEX”, Addison-Wesley, 1986.

[6] Silvio Levy and Donald E. Knuth: “CWEB User Manual: The CWEB System of Structured Documentation”, *Stanford University Research Report*, No.STAN-CS-90-1336, 1990.

[7] Wayne Sewell: “How to MANGLE your software: the WEB system for Modula-2”, *TUGboat*, Vol.8, No.2, pp.118–128, 1987.

[8] Norman Ramsey: “A SPIDER user's guide”, *Technical reports, Princeton University*, September 1988.

[9] John D. Ramsdell: “SchemeTEX”, Online file at prep.ai.mit.edu, The MITRE Corporation, 1990.

[10] Guy L. Steele Jr. et al.: “Common Lisp: the Language (second edition)”, Digital Press, 1990.

## 付録１：LispWEB(WEAVE) をかけて読みやすくした出力例

26. プログラム部分で defun 式により定義されている関数が、どのモジュールで使用されているかを表示する関数。

定義されている関数名のリストは、global 変数 *define-function-names* に (symbol-name . print-string) の形で保持されている。 その各関数名について、それが使用されているモジュール番号を関数 get-function-used-module-numbers により (結果は逆順になっているので、reverse する必要あり) 調べる。

```
(defun print-function-used-module-numbers ()
  (when *define-function-names*27
    ;; defun で定義される関数がある場合は、その関数が使われているモ
    ;; ジュール番号を表示する。
    (dolist (func *define-function-names*27)
      ;; func は、(symbol-name . print-string) の形になっている。
      (let ((modules (reverse
                      (get-function-used-module-numbers17 (car func)))))
        (when modules
          (write-to-TeX-file60 #\Newline "{\\footnotesize␣関数{\\bf␣" (cdr func))
          (write-to-TeX-file60 "} は、セクション" (car modules))
          (dolist (num (cdr modules))
            (write-to-TeX-file60 ",␣" num))
          (write-to-TeX-file60 "で使われている。}\\par" #\Newline)))
      (setq *define-function-names*27 '()))))
```

関数 print-function-used-module-numbers は、セクション 25 で使われている。

## 付録２：LispWEB(WEAVE) をかける前のソースプログラム例

```
;;; プログラム部分で defun 式により定義されている関数が、どのモジュールで
;;; 使用されているかを表示する関数。
;;;
;;; 定義されている関数名のリストは、global 変数 *define-function-names* に
;;; \verb+(symbol-name . print-string)+ の形で保持されている。 その各関数名
;;; について、それが使用されているモジュール番号を関数
;;; get-function-used-module-numbers により (結果は逆順になっているので、
;;; reverse する必要あり) 調べる。

(defun print-function-used-module-numbers ()
  (when *define-function-names*
    ;; defun で定義される関数がある場合は、その関数が使われているモ
    ;; ジュール番号を表示する。
    (dolist (func *define-function-names*)
      ;; func は、(symbol-name . print-string) の形になっている。
      (let ((modules (reverse
                      (get-function-used-module-numbers (car func)))))
        (when modules
          (write-to-TeX-file #\Newline "{\\footnotesize 関数{\\bf " (cdr func))
          (write-to-TeX-file  "} は、セクション" (car modules))
          (dolist (num (cdr modules))
            (write-to-TeX-file ", " num))
          (write-to-TeX-file "で使われている。}\\par" #\Newline)))
      (setq *define-function-names* '()))))
```