Macrodown (ver. 0.9) の仕様

gfn

2015 年 4 月 1 日

1　概要

Macrodown（**マクロダウン**）は LaTeX や HTML などのマークアップ言語をラップするための軽量［要出典］マークアップ言語です。マクロ定義機能があり，可変長引数を扱うこともできます。OCaml で実装しており，ソースコードは

`http://github.com/gfngfn/Macrodown`

で公開しています。この文書も Macrodown を経由して pLaTeX2ε と dvipdfmx で組んでいます。

2　動機

今日，一般的な文書作成のために多用されているマークアップ言語としては HTML+CSS や LaTeX などが挙げられます。しかし，この HTML+CSS と LaTeX は一長一短です。

HTML+CSS は構造と体裁がよく分離されており，CSS による体裁の変更等も柔軟ですが，記法が冗長で，またマクロ定義機能も存在していない[*1]ため，直接人間の手で文書を記述するには適しているとは思えません。Markdown，AsciiDoc，その他ローカルな Wiki 記法など，HTML を専用にラップする軽量マークアップ言語も数多くありますが，用途が限られている傾向にあり，表現力は乏しいと言わざるを得ないでしょう。

一方の LaTeX は比較的平易な記法でマクロ定義機能が備わっており，先達による高度なパッケージも充実し，そして何よりもソフトウェアとしての TeX が高品質な組版結果を出力してくれますが，構造と体裁が明示的に分離されていないため体裁を扱うには或る程度のリテラシーが必要であったり，柔軟なマクロを創るには TeX on LaTeX の知識が必要で，これが相当非直感的な仕様であるため，大多数の人を寄せつけない様相を呈しています。

[*1] おそらく意図的に取り入れられていないのでしょう。

これらの欠点を解決し，また Web ページと PDF を共通の規格で記述できるようにすべく，Macrodown の制作に手をつけました。記法は主に LaTeX を参考にしており，「意味論が綺麗な LaTeX 的記法」を目指したといってもよいかもしれません。ただし，LaTeX をラップするための言語としては LaTeX に記法が似ているということは利便性の上で障壁となりやすいので，今後各トークンに対する文字の割り当てに大きな改訂を行なう可能性も充分あります。

言語仕様として特に重視したのは可変長引数の扱いです。現在はまだ簡素な最小限の姿に留まっていますが，今後も条件分岐や整数を扱えるようにしたりといった拡張が考えられます。

3　仕様

3.1　トークン化

大小ラテン文字とアラビア数字とハイフンを**基本文字**，その他の文字を**非基本文字**として，Macrodown は文字列を以下にマッチするようにトークン化します。ただし * は 0 回以上の繰り返しを表します。複数通りのマッチがある場合はより長い文字列ほど優先し，同一の文字列が複数のトークンにマッチする場合は表で上位に掲げられているものほど優先されます。

`\macro`	**macro** ：マクロ定義トークン
`\pop`	**pop** ：ポップ機能トークン
`\`（基本文字）*	**cs** ：制御綴トークン
`@`（基本文字）*	**macro** ：変数名トークン
`#`（基本文字）*	**id** ：ID 名トークン
`{`	**{** ：開きトークン
`}`	**}** ：閉じトークン
`;`	**;** ：無引数終端トークン
`|`	**|** ：区切りトークン
`\`（非基本文字）	**char** ：エスケープされた文字トークン
（1 文字）	**char** ：文字トークン

ただし，このうち ID 名トークンを扱う操作は近い将来実装豫定なので，現在のところ ID 名トークンには用途がありません。

3.2　空白類を無視する条件

空白文字，タブ文字，改行文字を**空白類**と呼ぶことにします。空白類は無視されるものと文字トークンとされるものが出現場所によって決まり，以下に出現した場合は無視されます。

- 行頭
- 制御綴トークンの直後
- 変数名トークンの直後
- ID 名トークンの直後
- 空白類の直後

3.3 構文木化

eoi を入力の終端として，Macrodown は以下の T で表現される入力を受理します。

$$
\begin{aligned}
&T \to B\ \mathbf{eoi}\\
&B \to \varepsilon \mid SB\\
&S \to \mathbf{char} \mid \mathbf{var}\ ; \mid \mathbf{pop}\ \mathbf{var}\ \mathbf{var}\ GG \mid \mathbf{macro}\ \mathbf{cs}\ AG \mid \mathbf{cs}\ ; \mid \mathbf{cs}\ GP\\
&A \to \varepsilon \mid \mathbf{var}\ A\\
&P \to \varepsilon \mid GP\\
&G \to \{\ L\ \}\\
&L \to B \mid B \mid L
\end{aligned}
$$

3.4 変数とマクロ定義の基本

変数にはトークン列[*2]のみが格納され，現時点では数値などは扱えません。破壊的代入の機能はなく，したがって純粋函数型言語に近い性質をしています。

マークアップ言語における柔軟性を確保するため，前述のとおりマクロ定義機能があり，**定義式**

```
\macro \hoge @var1 ... @varN { Expr }
```

で引数を N 個伴うマクロ`\hoge` が定義されます。この定義式以降，**マクロ適用**として

```
\hoge {Arg1}... {ArgN}
```

で各 *varI* に *ArgI* が格納された上で定義式の *Expr* が評価されます。変数`@piyo` に格納された中身を取り出すには`@piyo ;` とします。例えば

[*2] 正確には構文木が格納されているのですが，トークン列が格納されていると見なしてもそれほど理解に齟齬のない仕様になっています。

```
\macro \test @x @y {Ma@x;r@y;down}%
\test{c}{o} \test{}{k}
```

に対する出力は

```
Macrodown Markdown
```

となります。引数のないマクロも定義でき，これは使用時にセミコロンをつけます。例えば

```
\macro \mcrd {Macrodown}%
Hello, \mcrd;!
```

に対する出力は

```
Hello, Macrodown!
```

となります。

3.5　可変長引数の扱い方

マクロ適用の引数には**可変長リスト**を使用することができます。可変長リストの形の引数を**可変長引数**と呼び，

```
{ Expr1 | Expr2 | ... | ExprN }
```

と記述します。マクロ適用時には，可変長リストはひとつの変数に格納されます。この可変長リストを 1 つづつ切り離す機能が\pop を用いた**ポップ式**で，

```
\pop @head @tail { Elem1 | Elem2 | ... | ElemN }{ Expr }
```

とすると変数*@head* に *Elem1* が，変数*@tail* に *Elem2* | ... | *ElemN* が格納された上で *Expr* が評価されます。変数*@head*，*@tail* が有効なスコープは *Expr* 内だけであることに注意してください。

なお，可変長リストが 1 個の要素 *Elem1* だけからなるときは，*@head* に *Elem1* が，*@head* に**空列**が格納された上で *Expr* が評価され，可変長リストが 0 要素，つまり空列であるときは *Expr* が評価されず，ポップ式全体が空列を返します。

可変長リストを扱う例を見てみましょう。

```
\sep @list {%
  \pop @head @tail {@list;}{(@head;)\sep{@tail;}}%
}%
\sep{bd|gfn|\sep{me|ip}}
```

に対する出力は

```
(bd)(gfn)((me)(ip))
```

となります。*@list* が空列を格納している場合はポップ式が空列を返すだけなのでそれ以上再帰が生じずに停止するのです。このほか，空列かどうかを判定する条件分岐機能\ifempty があり，

```
\ifempty{ ExprB }{ ExprT }{ ExprF }
```

で *ExprB* が空列のときは *ExprT* を評価し，空列でない時は *ExprF* を評価します。これを用いると例えば

```
\macro \enum-comma @list {%
  \pop @head @tail {@list;}{%
    @head;\ifempty{@tail;}{}{, \enum-comma{@tail;}}%
  }%
}%
\enum-comma{un|deux|trois}
```

により

```
un, deux, trois
```

を得ることができます。また，2 つのトークン列が同一の出力となるかどうかを判定する条件分岐機能\ifsame は

```
\ifsame{ ExprA }{ ExprB }{ ExprT }{ ExprF }
```

という形で使い，*ExprA* と *ExprB* が同一出力となる場合は *ExprT* を，互いに異なる出力となる場合は *ExprF* を評価します。