SCMの利用方法 (v0201)

上條哲男 (上智大学 経済学部)

平成14 年 4 月 20 日

1 SCMについて

第2章においてSCMの起動と終了方法を、第3章でSCMの実行方法を、第4章でエディタによるプログラムの作成と保存方法を、そして第5章でデータ編集機能を説明している。

2 SCMの起動・終了方法

2.1 SCMの起動方法

SCMを起動する方法は2つある。

第1の方法は、「スタート」をクリックし、「プログラム」を選択することから起動する方法で、第2の方法は、マイコンピュータをダブルクリックし、SCMがインストールされているドライブをダブルクリックすることから起動する方法である。

第　の方法

(1) 「スタート」をクリックし、「プログラム」を選択する。「MS-DOSプロンプト」をクリックする

(2) 画面に

```
Microsoft(R) Windows 95
   (C)Copyright Microsoft Corp 1981-1996.
```

と表示される。

(3) つぎのように入力していくと、

```
C:\WINDOWS>cd..

C:\>cd scm

C:\SCM>
```

と表示されていく。ここで、

```
C:\SCM>
```

の後ろに、scm を入力すると、

```
C:\SCM>scm
SCM version 4e1, Copyright (C) 1990, 1991, 1992, 1993, 1994
Aubrey Jaffer.
SCM comes with ABSOLUTELY NO WARRANTY; for details type `(terms)'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `(terms)' for details.

SCM Turtlegraphics Copyright (C) 1992 sjm@cc.tut.fi, jtl@cc.tut.fi
Type `(help-gr)' or `(help-turtlegr)' for a quick reference of
the new primitives.
;loading "berkeley.scm"
>
```

と表示される。

(4) 記号 >が促進記号で、ここから SCM が実行できる。

第　の方法

(1) Windows 95 のデスクトップにおいて、マイコンピュータをダブルクリックする。

(2) SCMがインストールされているドライブ (例: Windows _ 95(C:)) をダブルクリックする。
(3) SCMがインストールされているフォルダ (例: Scm) をダブルクリックする。実行するためのファイル (例: SCM.exe) をダブルクリックする。
(4) SCM が起動される。起動画面は、第1の方法と同じである。
(5) 記号 >が促進記号で、ここからSCMが実行できる。

2.2 SCMの終了方法

SCMを終了するには、

```
> (exit)
```

と入力する。

3 SCMの実行方法

SCMの実行は、利用者とSCM インタプリタとの間の会話形式でなされる。SCMを起動すると、会話が始まり、つぎのような促進記号（prompt,プロンプト）>が行の先頭に表示される（なお、SCMはSchemeの1種であり、SchemeはLISPの方言の1つである。ここでの説明は、他のSchemeにおいてもほとんど妥当する）。

```
>
```

なお、画面が全画面になっている場合に、それを編集画面にするには、ALTキーを押したまま、Enterキーを押す。その際に、画面に矢印が表示されてない場合には、画面を縮小し、幅を狭くする。

3.1では、SCMの基本的なインタプリタ機能を説明する。3.2では、作業の保存方法を、3.3では、データ・リスト用プログラムを直接作成する方法を、3.4では、ファイルを保存する方法を、そして3.5では、ファイルをロード (load) し、チェックする方法を説明する。

3.1 SCM の基本的なインタプリタ機能

促進記号 >の後ろに式を入力すれば、インタプリタはその式を評価し、結果を表示する。例えば促進記号の後ろに数値を入力しリターンキーを押すと、その数値が次の行に表示され、その次の行に促進記号が表示される。

```
> 1
1
>
```

数値演算は演算を示す記号と数値との組み合わせ、(+ 1 2) のような複合式 (compound expression) で行われる。

```
> (+ 1 2)
3
>
```

この式 (+ 1 2) は、数値の 1 と 2 とを加える、という意味である。演算子 (operator、オペレータ) をオペランド (operands) の前に置く、このような記法を前置記法 (prefix notation) という。この記法は、標準的な数学の記法と異なるため、最初はわかりにくいかもしれない。しかし、この記法にはいくつかの利点がある。その 1 つは、オペランドがいくつでもよいことである。例えば、

```
> (+ 1 2 3)
6
> (* 2 3 4)
24
>
```

などである。　いくつかの他の基本的なデータ形式はそれ自身で評価される。これらには、論理値も含まれる。

```
> t  ;;;true
#t

> nil  ;;;false
()
```

なお、セミコロン；は、注釈文字であり、セミコロン; の後ろは改行まで注釈となり、インタプリタから無視される。　文字列は、前後を引用符“ で囲む。

```
> "Statistics is an important subject."
"Statistics is an important subject."
```

インタプリタに記号 (Symbol) x が与えられたならば、その記号が値と関連づけられているかがチェックされ、関連づけられていれば、その値が表示される。与えられた記号が値と関連づけられていない場合には、次のようにエラーが表示される。

```
> x

ERROR: unbound variable:  x
; in expression: x
; in top level environment.
```

この時点では、記号 x はまだ値と関連づけられていない。記号 x に、例えば、値１を関連づけると、

```
> (define x 1)
x
```

となる。ここで、x を入力すると、

```
> x
1
```

となり、記号 x が数値１と関連付けられていることがわかる。記号名を入力するときは大文字でも、小文字でも使用できる。

複合式 (compound expression) の評価は少し込み入っている。複合式はスペース (空白記号、タブ記号、あるいは改行記号) で区切られ、括弧で囲まれたリストである。リストの各構成要素は文字列か、数値か、あるいは他の複合式かである。次のような複合式

```
(+ 1 2 3)
```

を評価するとき、インタプリタはこのリストを関数呼び出し (function application) を表現しているものとして取り扱う。リストの最初の要素は SCM の関数を表わし、その他の各要素を引数として適用し、結果を戻す。先の例では、関数は記号 +で表現される加算関数である。リストの残りの要素は引数である。引数は 1 と 2 と 3 である。この式を評価するように要求されると、インタプリタは、加算演算を引数に適用し、

```
> (+ 1 2 3)
6
```

という結果を返す。関数の引数はその関数を適用する前に評価される。上の式では引数はすべて値であり、それ自身が評価されたものである。しかし、

```
> (+ (* 2 3) 4)
10
```

では、インタプリタはまず引数 (* 2 3) を評価し、その後、関数 + を適用する。数値、文字、そして記号は SCM で使用できる基本的なデータ型である。これらの基本的なデータ型は「単純データ (simple data)」として表現できる。SCM では、「複合データ (compound data)」と呼ぶもっと複雑なデータも取り扱える。複合データの最も基本的な形式はリストである。リストは list 関数で作られる。

```
> (list 1 2 3 4 5)
(1 2 3 4 5)
```

インタプリタで表示された結果は複合式に似ている。すなわち、一連の要素がスペースで区切られ、括弧で囲まれている。SCM の式は、単純な list である。なお、SCM は Scheme の 1 種であり、Scheme は LISP の方言の 1 つである。LISP は LISt Processing に対する頭字語である。LISP は、数式の微分を数値計算す

るのではなく、数式そのものとして処理するというような、記号処理のタスクのためのツールとして、もともと開発された。

記号 y にリスト (1 2 3 4 5) を関連付けると、

```
> (define y (list 1 2 3 4 5))
y
```

となる。ここで、y を入力すると、

```
> y
(1 2 3 4 5)
```

となり、記号 y がリスト (1 2 3 4 5) と関連付けられていることがわかる。

リスト以外にもベクトルや多次元配列のような、種々の複合データがある。

インタプリタに (+ 1 2) というリスト自身を結果として戻し、表示させるにはどうしたら良いのだろうか。このままタイプしてインタプリタにわたしてしまうと、インタプリタはそのリストを解釈し、次のような結果を戻すことになる。

```
> (+ 1 2)
3
```

インタプリタにリストを評価させないように伝える方法は、「引用 (quoting)」という方法である。式そのものを扱うのに、式を引用符で囲むが、それと同じようなことである。例えば、引用符を使用した次の２つの文章の意味は非常に異なっている。

```
Say your hobby!        (あなたの趣味を言いなさい)
Say "your hobby"!      (  「あなたの趣味」と言いなさい)
```

SCM で引用 (quote) するためには次のようにする。

```
> (quote (+ 1 2))
(+ 1 2)
```

quoteは関数ではない。したがって、quoteは、関数を評価する、というすでに述べた規則には従わない、すなわち、その引数は評価されない。quoteは特別形式 (special form) と呼ばれる。特別という意味はその引数の取り扱いの規則が特別であることを意味している。その他の特別形式は必要なところで説明する。上述の基本的な評価規則と特別形式とで、SCM 言語のシンタックスができている。　特別形式 quote に関しては、引用符を式の前につける簡便的な記法がしばしば使用される。

```
> '(+ 1 2)
(+ 1 2)
```

これは (quote (+ 1 2)) と同じである。この引用符は前だけで、後ろには必要ない。

3.2　作業の保存方法

ユーザーとインタプリタとの間でのセッションを記録したい場合には、transcript-on 関数を使用する。フロッピードライブ a: の フォルダ（ディレクトリ）SCM-tk 内のフォルダ tran1999 内に記録する。ここで、tk は著者 (上條哲男）の氏名のイニシャルであり、tran は transcript を、そして 1999 は西暦 1999 年を示している。ファイル名を mmddhhmm とする（ファイル名に拡張子は付けない)。ここで、mm:は月、 dd:は日、 hh:は時、 そして mm:は分である。なお、同じファイル名を使用すると、上書きされてしまい、以前の内容は消去されてしまうので、注意する必要がある。フォルダ名およびファイル名の付け方は、MS-DOS の約束に従うものとする。したがって、使用できる文字数は 8 文字以内である。また、ファイルの拡張子は 3 文字以内である。

transcript-on 関数を使用するために、まず、フロッピーディスクにフォルダ (ディレクトリ）SCM-tk を、さらにその内にフォルダ tran1999 を作成しておくことにする。1999 年の 9 月 29 日の 17 時 50 分に transcript-on を開始するものとすると、

```
(transcript-on "a:/SCM-tk/tran1999/09291750")
```

とする。transcript を終了するには、

```
> (transcript-off)
```

とする。なお、途中で transcript のためのファイル、

```
"a:/SCM-tk/tran1999/09291750"
```

の内容を見るためには、まず、

```
> (transcript-off)
```

として、transcript を終了させておく必要がある。あらためて セッションを記録するためには、再度 transcript-on を実行する必要がある。

3.3 データ・リスト用プログラムを直接作成する方法

60 回サイコロを投げる、というサイコロ投げの実験を実施し、第 1 回から第 30 回までの実験データのためのプログラムを SCM システムで直接作成する方法を説明する。そのデータ・リスト名を sr0130tk とする。、ここで、

```
sr:   s: サイコロ;  r: red(赤)
0130: 実験: 第 1 回から第 30 回まで
tk:   t: tetsuo,  k: kamijo
```

である。なお、データ・リスト名の付け方は、MS-DOS の約束に従う。したがって、使用できる文字数は 8 文字以内である。まず、サイコロ投げの実験データ用のプログラムを SCM に入力する。

```
> (define sr0130tk (list
     6 2 1 3 5 5 4 5 2 3
     2 6 4 5 6 5 5 4 5 6
     1 1 6 1 5 4 5 4 5 5))
sr0130tk
```

各データが正しく入力されているかどうかをチェックするために、データを表示してみることにする。

```
> sr0130tk
(6 2 1 3 5 5 4 5 2 3 2 6 4 5 6 5 5 4 5 6 1 1 6 1 5 4 5 4 5 5)
```

さらに、合計を計算すると、

```
> (apply + sr0130tk)
121
```

となる。

3.4 ファイルの保存方法

ファイルを、フロッピードライブ a:のフォルダ SCM-tk 内に保存するものとする。

(1) SCM 内のデータ・リスト用プログラムを選択し、「コピー」をクリックする。ここでは、プログラムは、

```
(define sr0130tk (list
   6 2 1 3 5 5 4 5 2 3
   2 6 4 5 6 5 5 4 5 6
   1 1 6 1 5 4 5 4 5 5))
```

である。

(2) エディタ（例えば、秀丸、メモ帳、WZ、あるいは MIFES など。なお、ここでは、メモ帳を使用することにする）を起動し、「貼り付け」をクリックする。
(3)「ファイル」をクリックし、次に「名前を付けて保存」をクリックする（なお、「上書き保存」でよい場合もある)。「ファイルの種類」を「すべてのファイル」とする。「保存する場所」をフロッピードライブ a:のフォルダ SCM-tk とする。ファイル名を tkamijo.scm として、保存」をクリックする。。

3.5 ファイルをロードし、データをチェックする方法

ファイルおよびデータ・リストをロード (load) する方法を、3.4 においてフロッピードライブ a: のフォルダ SCM-tk 内のファイル tkamijo.scm に保存したデータ・リスト sr0130tk を用いて説明することにする。

(1) SCM システムにロードするのと同じデータ・リストがあるかどうかをチェックし, あればそれを削除する。ここでは、ロードするデータ・リスト sr0130tk があるので、それを削除する。念のため削除されているかどうかをチェックする。

```
> sr0130tk
(6 2 1 3 5 5 4 5 2 3 2 6 4 5 6 5 5 4 5 6 1 1 6 1 5 4 5 4 5 5)
> (define sr0130tk nil)
sr0130tk
> sr0130tk
()
```

(2) SCM システムにおいて、フロッピードライブ a: のフォルダ SCM-tk 内のファイル tkamijo.scm およびその内のデータ・リスト sr0130tk をロードする。

```
> (load "a:/SCM-tk/tkamijo.scm")
okay
```

(3) データ・リスト sr0130tk がロードされたことをチェックする。

```
> sr0130tk
```

```
(6 2 1 3 5 5 4 5 2 3 2 6 4 5 6 5 5 4 5 6 1 1 6 1 5 4 5 4 5 5)
```

確かに、データ・リスト sr0130tk がロード されている。

ここで、sr0130tk というリスト名 (list name) は長いので、s0130(saikoro, 01 回から 30 回まで) と定義し直すことにする。

```
> (define s0130 sr0130tk)
s0130
```

とする。再度、s0130 を確認すると、

```
> s0130
(6 2 1 3 5 5 4 5 2 3 2 6 4 5 6 5 5 4 5 6 1 1 6 1 5 4 5 4 5 5)
```

となっていることがわかる。以降、データ・リスト sr0130tk に関する演算は データ・リスト s0130 に関する演算として実行できることになる。

4 エディタによるプログラムの作成・保存方法

エディタでデータ・リスト用のプログラムを作成し、そのプログラムをフロッピーディスクに保存する方法を説明する。なお、エディタとして Windows 95 のメモ帳を使用することにする。また、スプレッドシート用のアプリケーションである Excel や Lotus 1-2-3 によってファイルを作成し、それをロード することも可能であるが、その説明は割愛する。

フロッピードライブ a:のフォルダ SCM-tk 内のファイル tkamijo.scm にデータ・リスト用のプログラムを作成する。60 回サイコロを投げる、という実験の第 31 回から第 60 回までの結果をデータ・リストとし、データ・リスト名を、sr3160tk とする。なお、データ リスト名の付け方は、MS-DOS の約束に従うものとする。したがって、使用できる文字数は 8 文字以内である。

(1) フロッピードライブ a: にフロッピーディスクを入れる。

(2) Windows95 のアクセサリ内のメモ帳を使用して、データ・リスト用のプログラムを作成する。Windows95 のデスクトップにおいて、スタートをクリックする。プログラム、アクセサリ、そしてメモ帳を順次選択し、順次クリックする。メモ帳 が開く。

(3) フォルダ SCM-tk 内のファイル tkamijo.scm にデータ・リスト用のプログラムを作成するという具体例を示す。なお、" ; "以下は注釈であり、プログラムの実行には全く影響を与えない。また、SCM では変数名に日本語を使用できない。メモ帳においてファイル (F) をクリックし、「開く」をクリックする。ファイルの種類は、すべてのファイル (*.*) を選択する。フロッピードライブ a:をダブルクリックし、フォルダ SCM-tk をダブルクリックする。tkamijo.scm をクリックし、「開く」をクリックする。すでに、

```
(define sr0130tk (list
   6 2 1 3 5 5 4 5 2 3
   2 6 4 5 6 5 5 4 5 6
   1 1 6 1 5 4 5 4 5 5))
```

が入力されているはずである。このプログラムのつぎに、

```
;;;  file name: tkamijo.scm
;;;             tkamijo:  tetsuo kamijo
;;;             scm:      Scheme 言語
;;;             作成者:   上條哲男
;;;             作成開始: October 5, 1999
;;;             改訂:
;;;
;;;  data list name:  sr3160tk
;;;        sr:        s: saikoro;  r: red(赤)
;;;        3160:      実験:  第 31 回から第 60 回まで
;;;
;;;        作成者:    上條哲男
;;;        作成開始:  October 5, 1999
;;;        改訂:
;;;
     (define  sr3160tk (list
              3 1 1 2 6 4 2 6 4 3
              3 5 1 3 1 1 4 4 1 5
              3 2 6 2 3 6 1 1 2 1) )
```

と入力する。

(4) ファイル tkamijo.scm を保存する。まず、ファイル (F) をクリックし、「上書き保存」をクリックする。

ファイルをロードし、データをチェックする方法は、3.5 で説明したが、再度説明することにする。

フロッピードライブ a:のフォルダ SCM-tk 内に保存しファイル tkamijo.scm およびファイル tkamijo.scm 内のたデータ・リスト sr3160tk をロード (load) する。

(1) SCM システムにロードするデータ・リストと同じものがあるかどうかをチェックし, あればそれを削除する。
(2) SCM システムにおいて、tkamijo.scm をロードする。

```
> (load "a:/SCM-tk/tkamijo.scm")
okay
```

(3) そのファイル内のデータ・リスト sr3160tk をチェックすると、

```
> sr3160tk
(3 1 1 2 6 4 2 6 4 3 3 5 1 3 1 1 4 4 1 5 3 2 6 2 3 6 1 1 2 1)
```

となっている。
さらに、合計を計算すると、

```
> (apply + sr3160tk)
87
```

となる。

sr3160tk というリスト名 (list name) は長いので、s3160 (saikoro, 31 回から 60 回まで) と定義し直すために、

```
> (define s3160 sr3160tk)
s3160
```

とする。再度、s3160 を確認すると、

```
> s3160
(3 1 1 2 6 4 2 6 4 3 3 5 1 3 1 1 4 4 1 5 3 2 6 2 3 6 1 1 2 1)
```

となっていることがわかる。以降、データ・リスト sr3160tk に関する演算は データ・リスト s3160 に関する演算として実行できることになる。

5 データ編集機能

データ編集機能として、リストの結合およびデータの変更（修正）について説明する。

5.1 リストの結合

複数の短いリストをまとめて１つの長いリストにしたい場合は、関数append を使う。例えば、

```
> (define x1 (list 1 2 3))
x1
> (define x2 (list 4))
x2
> (define x3 (list 5 6))
x3
```

のとき、つぎのようになる。

```
> (append x1 x2 x3)
(1 2 3 4 5 6)
```

なお、append の引数はリストでなければならない。

ここで、データ・リスト sr0130tk と sr3160tk とを、append し、append されたデータリストを sr0160tk(saikoro, 01 回から 60 回まで）と定義すると、

```
> (define sr0160tk (append sr0130tk sr3160tk))
sr0160tk
```

となる。sr0130tk をチェックすると、

```
> sr0160tk
(6 2 1 3 5 5 4 5 2 3 2 6 4 5 6 5 5 4 5 6 1 1 6 1 5 4 5 4 5 5
 3 1 1 2 6 4 2 6 4 3 3 5 1 3 1 1 4 4 1 5 3 2 6 2 3 6 1 1 2 1)
```

となっている。
さらに、合計を計算すると、

```
> (apply + sr0160tk)
208
```

となる。
sr0160tk というリスト名 (list name) は長いので、s0160 (saikoro, 01 回から 60 回まで) と定義し直すために、

```
> (define s0160 sr0160tk)
s0160
```

とする。再度、s0160 を確認すると、

```
> s0160
(6 2 1 3 5 5 4 5 2 3 2 6 4 5 6 5 5 4 5 6 1 1 6 1 5 4 5 4 5 5
 3 1 1 2 6 4 2 6 4 3 3 5 1 3 1 1 4 4 1 5 3 2 6 2 3 6 1 1 2 1)
```

となっていることがわかる。以降、データ・リスト sr0160tk に関する演算は データ・リスト s0160 に関する演算として実行できることになる。

5.2 データの変更（修正）

特別形式 (special form) set! は、変数の値を変更するために使われる。変数 z を次のように定義しておく。

```
> (define z 2)
```

ここで、z の値を 5 に変更するには、

```
> (set! z 5)
```

とする。

参考文献

[1] Harvey, B. and Wright, M.[1994], Simply Scheme: Introducing Computer Science, The MIT Press.

[2] 猪俣俊光、益崎真治 [1994], Schemeによる記号処理入門，森北出版.

[3] Pearce, J.[1998], Programming and Meta-Programming in Scheme, Springer.

[4] Springer, G. and Friedman, D.P.[1989], Scheme and the Art of Programming, The MIT Press.

[5] Steele, Jr., G.L.[1990], Common LISP: The Language, Second Edition, Digital Press.

[6] Tierney, L.[1990], LISP-STAT: An Object-Oriented Environment for Statistical Computing andf Dynamic Graphics, Wiley.
邦訳: 垂水共之，鎌倉稔成，林　篤祐，奥村晴彦　水田正弘　共訳，LISP-STAT, 共立出版.

[7] 湯浅太一 [1991], 『Schemeマニュアル』, 岩波書店.

[8] 湯浅太一 [1991], 『Scheme入門』, 岩波書店.