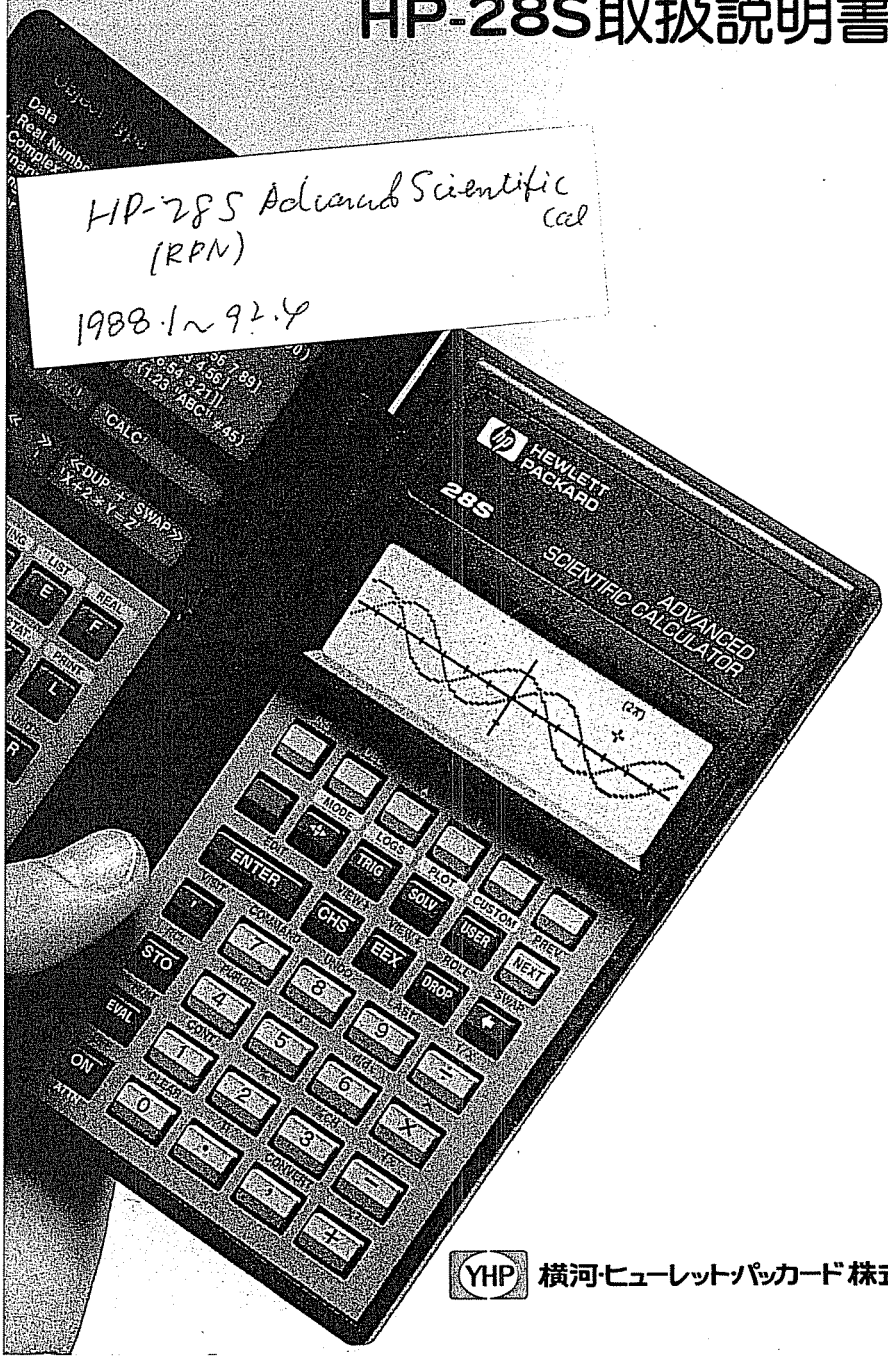
HEWLETT-

HP-28S取扱説明書

HP-28S Advanced Scientific cal
(RPN)
1988.1～92.9



YHP 横河・ヒューレット・パッカード株式会社

HP-28S取扱説明書

YHP 横河・ヒューレット・パッカード株式会社

お知らせ

この説明書はアメリカのHewlett-Packard(ヒューレット・パッカード)社のHP-28S Advanced Scientific Calculator Owner's Manualの初版(Nov, 1987)を基にして作成しました。
この製品の保証と規制情報については、291ページと295ページを見てください。
この説明書に記載した内容を予告なく変更することがあります。**当社を含むヒューレット・パッカード社グループはこの説明書に記載した内容が商用または特殊目的に適合するかしないかについては保証いたしかねます。**ヒューレット・パッカード社グループはこの説明書に記載したエラーや製品の内容性能、この説明書の内容を組み合わせたことによる直接的または間接的損害については責任を負いかねます。ヒューレット・パッカード社グループはヒューレット・パッカード社が組み込んだソフトウェア以外のソフトウェアの使用や信頼性については責任を持ちかねます。

Hewlett-Packard Company
Corvallis Division
1000 N.E. Circle Blvd.
Corvallis, Oregon 97330, U.S.A

〒168 東京都杉並区
高井戸東三丁目29番21号
横河・ヒューレット・パッカード株式会社

HP-28Sについて

HP-28Sをお買い上げくださいましてありがとうございました。HP-28Sを使うと、従来の小型計算機では望むことができなかった問題を含めて、かなり複雑な問題も簡単に解くことができます。HP-28Sは強力な数値計算機能と、新しい機能(定数と変数が入っている代数式をそのまま直接解くこと)を組み合わせたものです。a、b、cのような定数やx、y、zのような変数を、英字の名称で表すことで目的の問題を数式化して、その一般的な性質を表した名称が入ったままの答えを求めたり、数値化した答えを求めることができます。

HP-28Sには次のような機能があります。

- 数式処理　代数式の括弧を開いて、同類項をまとめるなどの操作ができます。また等式を任意の変数について代数式のまま解くことができます。
- 微積分　導関数、不定積分が計算できます。
- 数値　HP-28SのSOLVEを使うと、変数が多数入っている数式や等式でも解くことができます。連立一次方程式を解くこともできます。各種のデータ型に対応しているので、複素数、ベクトル、行列でも、実数と同じように簡単に扱うことができます。
- プロット　数式、等式、統計データをグラフ化することができます。
- 単位換算　120種もの計数単位を内蔵しているので、簡単に単位の換算ができます。任意の単位を定義することもできます。
- 統計計算　1変量統計、2変量統計、統計確率が計算できます。
- ビット整数演算　コンピュータなどで使っているビット整数を、2進数、8進数、10進数、16進数のどれかで表したもののどんな組み合わせでも計算できます。
- 算式通り入力の方法と、対話型計算ができる逆ポーランド法の両方を、任意に組み合わせることができます。

HP-28S取扱説明書(本書)は3部に分けました。第1部の「基本操作」では、簡単な問題をいくつか取り上げて、HP-28Sの使用法を説明いたします。第2部の「基本的機能の要点」では、第1部の問題を皆様の問題に展開するためのヒントです。第3部の「プログラミング」では、プログラムの機能を説明し、20種のプログラム例で機能の利用法をお目にかけます。

HP-28S Reference Manualは各コマンドの詳細説明です。メニューごとの辞書になっていて、各メニュー内のコマンドや考え方を説明します。

HP-28Sの操作に慣れるために、まず第1部の例題通りに操作することをお勧めいたします。次に第2部を読んで、操作法を広く理解していただくようお願いいたします。特定のコマンドを詳しく知りたいときには、Reference Manualを調べてください。プログラムについて知りたくなりましたら、第3部の「プログラミング」を見てください。

2冊の説明書で数学的な問題にHP-28Sをどう使うか取り上げましたが、数学そのものについては各種の参考書をご利用くださるようお願いいたします。皆様が当面必要な部分についてだけ、この説明書に目を通してください。

御注意

皆様がプログラムなどをお作りになりましたら、そのプログラム内容と、試験用の数値例をノートに記録しておくことをお勧めいたします。また大切な各種計算用データも同様に記録しておくことをお勧めいたします。HP-28Sの不揮発性メモリ内がいっぱいになると、どれかを削除しないと次に進めなくなります。また電池の寿命切れや、HP-28Sの不適当な操作や修理などで、記憶内容が消えてしまうことがあります。そのようなときでも、記録があればいつでも再現できます。HP 82240Aプリンタをご利用いただきますと、簡単に記録が取れます。

目次

第２部　基本的機能の要約

第3部　プログラミング

付録と索引

この説明書のじょうずな利用法

この説明書の始めから最後まで例題全部を試してみるのが最上の方法です。もしその時間がないときはとりあえず次のようにして始めてください。

1. 第1部「基本操作」の初めの5章を読んで計算機に慣れてください。
2. HP-28Sとは異なる種類の計算機との比較が付録ページにあります。
   - HP社製のRPN 方式計算機に慣れている方は296ページ付録B「HP社製RPN 計算機に慣れている方への注意事項」を読んでください。
   - [=]キーを使うタイプの計算機に慣れている方は、302ページの付録C「算式通り入力の計算機に慣れている方への注意事項」を読んでください。
3. 第1部の残りの部分に載っているもののなかから関心のあるものだけ選んでその章の例題を試してみてもよいでしょう。

この説明書の内容

第1部「基本操作」ではいくつか簡単な問題を解く方法を説明します。この例題に取り組んでいるうちにHP-28Sの操作方法や、オブジェクトの種類、そしてメニューについて覚えることができるようになります。

第2部「基本的機能の要約」は第1部を補強するものです。計算機の使い方について詳しく説明してあり、別の用法や第1部で触れていなかったことについて書いてあります。第2部を利用すれば第1部の例を拡張して自分の問題解決に計算機を使えるようになります。

第3部「プログラミング」はHP-28Sのプログラム機能の特長の説明です。最後の章である「プログラム例」にはプログラム上のコツがわかる短いプログラムや便利なプログラムがあります。

詳細情報は

この説明書の例題を解き進むにつれて例題中に見られるいろいろな点について疑問を持たれることがあると思います。この説明書とReference Manualにはもっと詳しいことが載っています。

- 問題点が生じたら282 ページ「Q&A集」を見てください。
- 各キーについて簡単に機能内容を知りたいときは327 ページ「キーの索引」を見てください。
- 各メニューのコマンドについて簡単に機能内容を知りたいときは306 ページ「メニュー案内」を見てください。
- ひとつのメニューについて詳しく知りたいときはReference Manualを参照してください。すべてのメニュー（といくつかの話題）がアルファベット順に出ています。ディクショナリ部の見出しはReference Manualの裏表紙にリストしてあります。
- 特定のコマンドについて詳しく知りたいときはReference Manualの終わりの〝Operation Index〟を参照してください。メニュー名とディクショナリ部の参照ページがすぐにわかります。

第1部

基本操作

第1章

初めて使う方へ

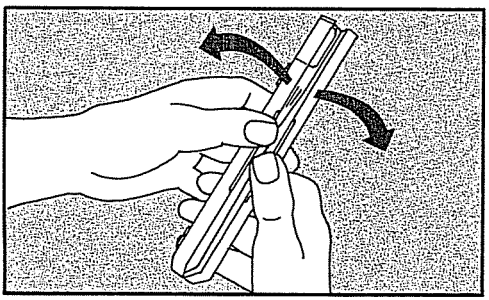
この章では最初にこの計算機の外観を説明し、それから簡単な計算例を取り上げます。次に、キーボードと表示部の主要部分を注釈付きの解説図で説明いたします。最後にコマンドのカタログ機能を説明しますが、これはコマンドの索引とコマンドの使用法の簡単な案内が出る機能です。

予備知識

ここでは計算機の外観などを説明します。

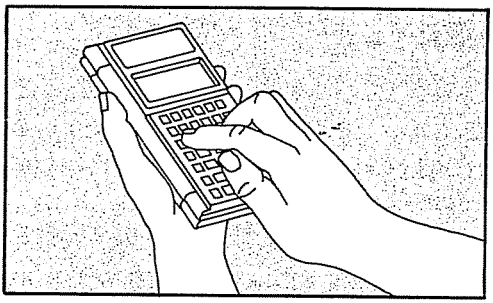
ケースの開閉

この計算機はプラスチック・ケースと一体になっていて、手帳のように開閉します。ケースを開くには、背中側を向こう側にして図のように開きます。



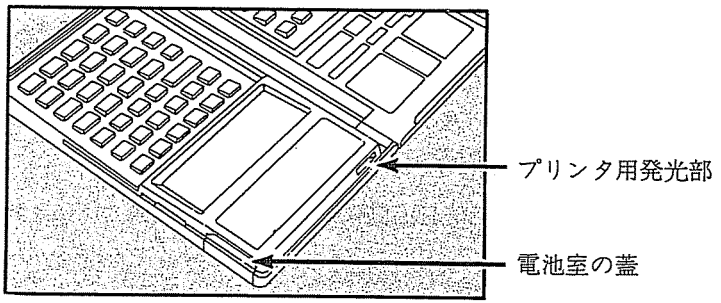
ケースを閉めるには、カクンと音がするまで両側を閉じてください。

計算機の左半分を右側の下にまで折り曲げることができます。これで片手で持って別の手で操作するような現場作業や、机の上が狭くても使えます。



電池室の蓋とプリンタ用発光部の位置

計算機を開くと、電池室の蓋とプリンタ用の赤外線発光部が見えます。



HP-28Sは3本の単5型アルカリ乾電池（日本ではAM-5型、アメリカではN型、ヨーロッパではLR1型）を使っています。電池は工場出荷時に取り付けて試験をしてあります。電池の交換法は286ページで説明します。同じ大きさのマンガン乾電池（こちらはUM-5）も使えますが、電池の寿命が短くなります。

HP-28Sでプリンタを操作するときには、計算機からプリンタに赤外線で情報を送っています。この信号はプリンタ用発光部から送り出されて、プリンタが受け取ると印字されます。プリンタの操作方法は149 ページからで説明します。

HP-28Sの電源スイッチ

[ON]キーを押すと電源スイッチが入ります。HP-28Sには不揮発性メモリを使っているので、スイッチを切ってもそれまでの全データと表示内容をそのまま記憶しています。

電源スイッチが入っているときには、[ON]キーの下側にATTN（attention、中断）と白で印刷してあるように動作を中断する働きがあります。[ON]を押すとキー入力した文字や数値が消えてプログラムも停止します。

▒[OFF]を押すとHP-28Sの電源スイッチが切れます。（"▒[OFF]を押す"とは〝シフト（機能切り替え）キーの▒を押して放し、次にキーの上側に赤色でOFFと印刷してあるキーを押す〟ことです。）

HP-28Sを使わないで約10分間放置すると、電池が長持ちするように自動的にスイッチが切れます。[ON]を押せばまたスイッチが入ります。

全メモリ内容のクリア（メモリのリセット）

メモリをリセットするとHP-28Sが工場出荷時の状態に戻ります。HP-28S内の全情報（プログラムやデータなど）が消えてしまいます。ユーザが変更したどの状態（数値表示形式や角度単位など）も初期値に戻ります。

メモリをリセットするには次の順に操作します。

1. [ON]を押し続けます。
2. [INS]（一番上の左端のキー）も押し続けます。
3. [▶]（一番上の右端のキー）を押し放します。
4. [INS]キーを放します。
5. [ON]キーも放します。

HP-28Sからピッと音が出て Memory Lost. の文字が出ます。この文字は次にどのキーを押しても消えます。

メモリのリセットを中止するには、[ON]を押し続けているときに[DEL]（[INS]の右隣り）を押して放し、次に[ON]も放します。[DEL]を押したときにリセット操作を取り消したことになります。

表示の濃さの調節

まわりの明るさや見る角度に応じて表示の濃淡を調節することができます。

濃淡を調節するには次の順に操作します。

1. [ON]を押し続けます。
2. [+]を何回か押すと表示が濃くなり、[−]を何回か押すと表示が淡くなります。
3. [ON]を放す。

簡単な計算

次の計算をやってみましょう。

$$(15+23)\times\sin 30^\circ$$

基本的な手順は筆算のときと同じです。まず15＋23を計算して、中間結果を出します。次にsin30°を計算して、別な中間結果にします。最後に、両方の中間結果を組み合わせて答えを出します。

数値のキー入力を途中で間違えたら、次のようにして訂正します。

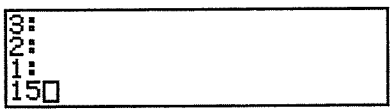
- [◀]を押すとキー入力した最後の数字が消えます。
- [ON]を押すとキー入力した全部の数字が消えます。

白紙の状態で計算を始めましょう。

[ON] [CLEAR]

表示部内に見えるのはスタックで、これが作業領域です。スタックはからになっています。
[1] [5] を押すとコマンド行に15を書き込んだことになります。

15

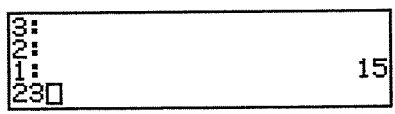


コマンド行の分だけスタックが上に動くので、スタックの3段分だけが見えます。
15をスタックに入れましょう。

[ENTER]

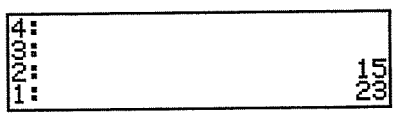
数値がスタックの1段目（左端に1：がある行）に入りました。コマンド行が消えて、スタック4段分がまたみえるようになりました。
[2] [3] を押してコマンド行に23を書き込みましょう。

23



23を1段目に入れましょう。

[ENTER]



1段目に入っていた数値の15が、2段目に上がりました。

15と23を足しましょう。

[+]

数値の15と23がスタックから消えて、その合計の38が1段目に返りました。この中間結果は2番目の中間結果を計算するまで、スタックに残しておきます。sin30°を計算するためにTRIG (trigonometry、三角法) メニューを使いましょう。

[TRIG]

表示部の一番下の行にTRIGメニューの6個のコマンドが見えます。この6個のメニュー表示（ SIN ～ ATAN ）で6個のメニュー・キー（表示部のすぐ下側のキー）の働きを定義しています。
[3] [0] を押してコマンド行に30を書き込みましょう。

30

30を1段目に入れます。

[ENTER]

前の結果（15+23=38）は2段目に上がりました。
sin30°を計算しましょう。

SIN

数値の30が1段目から消えて、その正弦値の .5が1段目に返りました。以前の結果の38は2段目に残っています。
38× .5を計算しましょう。

[×]

1段目と2段目の数値の38と .5が消えて、その積の19が1段目に返りました。
これで計算が終わりました。

$$(15+23)\times\sin 30^\circ=19$$

要約すると、上の計算の一般的手順は次のようになります。

1. コマンド行に数値を1個キー入力します。
2. [ENTER]を押して数値をスタックに入れます。
3. コマンドを実行するキーを押します。(キーボードにコマンドが見えないときには、そのコマンドが入っているメニューに切り替えて、適切なメニュー記号の下側のメニュー・キーを押します。)

上の例ではどの計算もスタックで実行しました。この方式の重要点は、[ENTER]を押してどの数値もスタックに入れたことです。実際には、続けてキー入力した2個の数値を分けるためにだけ[ENTER]を押すことが必要で、上の例では、15と23を分離しました。上の例の2番目と3番目の[ENTER]を省略して、もう一度計算してみてください。

数学関数を実行する前に数値にスタックを入れる上記の計算方式を、逆ポーランド記法（RPN）とか後置法、スタック方式と呼びます。HP-28Sの大部分のコマンドは、計算用でなくても、この方式を利用しています。この方式は2個の単純な規則を使っています。

- 関数が要求している入力を、関数の引き数と呼び、その関数を実行する前にスタックに入れておくことが必要です。
- 関数の結果はスタックに返り、これを次の関数の引き数として使うこともあります。

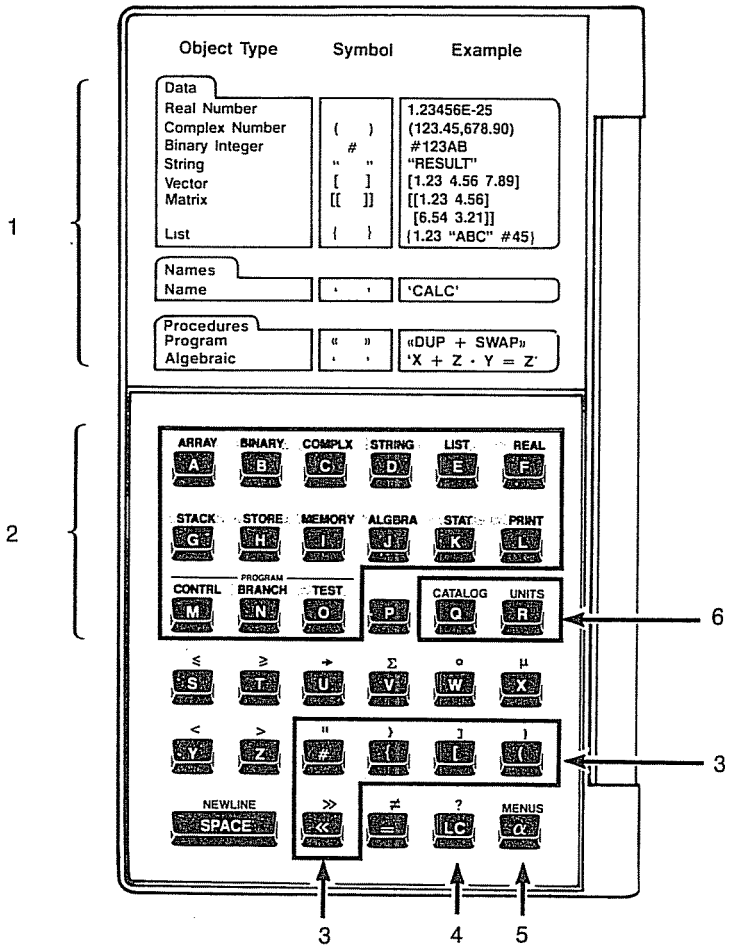
HP-28Sでは書物に記載してあるような、数式の形で表したものを入力して計算することもできます。34～35ページで数式を使って、上記と同じ計算をお目にかけます。

計算機の概要

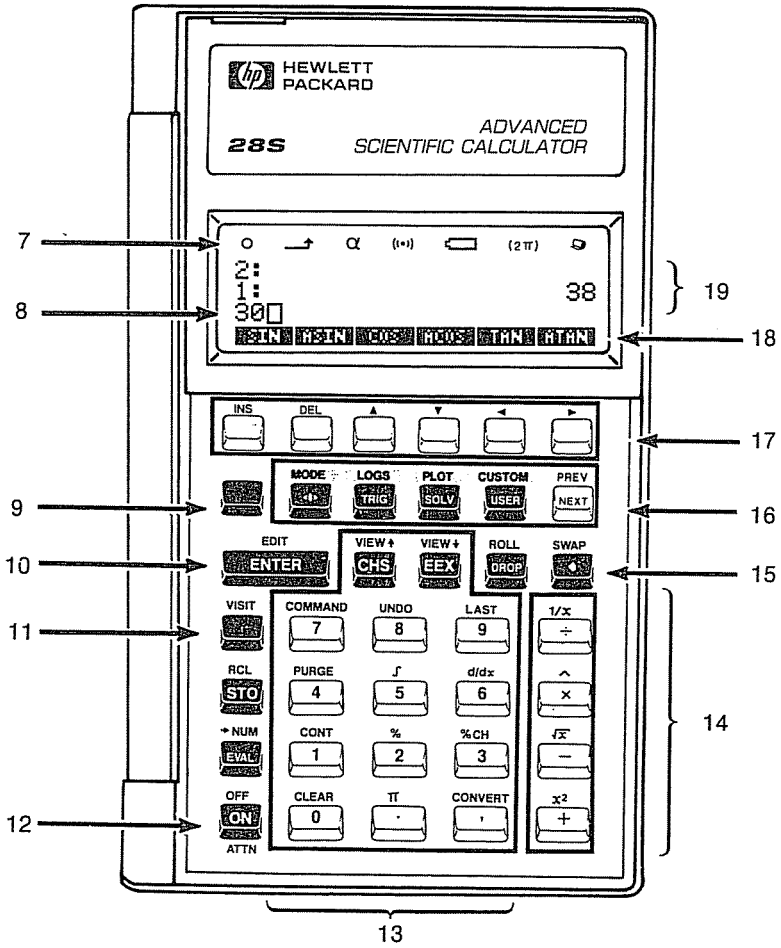
ここでは計算機の各部分の説明と、各コマンドとその使用法を見ることができるコマンドのカタログ機能を取り上げます。

主な機能と概念

26～27ページの図は計算機のキーボードと表示部で、重要な部分ごとに区分けしてあります。図内の番号は28ページからの番号に対応しています。



1．オブジェクトの型と形式
2．メニュー切り替えキー
　（シフト・キーを押した後）
3．オブジェクトの識別記号
4．小文字切り替え
5．入力モード
6．コマンドと単位のカタログ表示
　（シフト・キーを押した後）



7．案内表示
8．コマンド行
9．シフト・キー
10．コマンド行の処理
11．記号表現オブジェクトの識別記号
12．電源スイッチ、コマンド行の取り消し、プログラムの停止
13．数値入力
14．算術演算
15．1字取り消し
16．メニュー切り替え、次のメニュー行
17．メニュー・キー
18．メニュー記号
19．スタックの段の番目

1．オブジェクトの型と形式　「オブジェクト」とは計算機で処理する個々の対象を一括して言い表す用語です。この表には10種の基本的なオブジェクトの識別記号と例があります。

- 実数。5とか-4.3×10^{15}などです。
- 複素数。これは複素数x＋yiまたは平面の座標を表す2個1組の実数です。
- ビット整数。これはコンピュータ分野で使っている正負符号が付かない整数です。
- 文字列。これは任意の文字が続いたものです。
- ベクトル。これは線型代数で使っている一次元の配列です。
- 行列。これは線型代数で使っている二次元の配列です。
- リスト。オブジェクトを任意の順に並べたものです。
- 名称。他のオブジェクトに名前をつけて保存したり、記号計算ができるようにするもの。
- プログラム。ユーザ独自のコマンドや関数が作れます。
- 数式オブジェクト。数学の演算式や方程式を表したものです。

2．メニュー切り替えキー（シフト・キーを併用）　メニュー切り替えキーを押すとメニュー・キーにコマンドが割り当てられます。例えば、▒ [ARRAY] を押すとARRAYメニューに切り替わります。違うメニューに切り替えるには、目的のメニュー切り替えキーを押します。

右側のキーボードにも別のメニュー切り替えキーがあります（第16項参照）。

3．オブジェクト識別記号　この記号でオブジェクトの型を区別します（1参照）。例えば、[#] はビット整数を表し、[«] と▒[«] の間はプログラムを表します。

実数には識別記号が不要です。記号表現のオブジェクト（名称と数式）は右側のキーボードにある ['] 記号が必要です（第11項参照）。

4．小文字切り替えキー　小文字をキー入力するには [LC] を押します。小文字入力モードは [LC] をもう一度押すか、コマンド行を処理するために [ENTER] を押すか、コマンド行を取り消すために [ON] を押すまで続きます。

5．入力モード　コマンド行には3種の入力モードがあって、それぞれが特定のオブジェクト型の入力に適するようになっています。オブジェクトのキー入力で入力状態が自動的に変わりますが、時には手動で変えたいことがあります。[α]キーで入力モードを切り替えることができます。

6．コマンドと単位のカタログ機能（シフト・キーを併用）　[■] [CATALOG] を押すとHP−28Sの全コマンドとそれに必要な引き数を見ることができます(31ページ参照)。[■] [UNIT] を押すと単位換算に使える単位を見ることができます（141ページ参照)。

7．案内表示　これで計算機の各種の状態を表します。それぞれの案内表示が表示部に見えたら、次のことを表します。

案内表示	意　味
	プログラムが中断しています。
	シフト・キー([■])を押しました。
α	文字入力状態になっています。
((●))	作業中なので入力できません。
	電池の電圧が低下しています。
(2π)	角度がラジアン単位になっています。
	プリンタにデータを送っています。

8．コマンド行　キー入力した文字や数字はコマンド行に入ります。

9．シフト・キー　キーの上側に赤で印刷してあるコマンドを実行するには赤色のシフト・キー（[■]）を押します。

10．コマンド行の処理　コマンド行の内容を実行するには [ENTER] を押します。

11．記号表現オブジェクトの識別記号　識別記号はオブジェクトの型を区別するために前後に付ける記号で、記号表現オブジェクトは名称と数式オブジェクトのことです。記号表現オブジェクトをキー入力するには、オブジェクトの前と（必要に応じて）後に ['] を押します。

実数には識別記号が不要です。これ以外のオブジェクト識別記号は左側のキーボードにあります（1と3参照）。

12. 電源の入・切、コマンド行のクリア、プログラム実行の停止　計算機の電源を入れるには、[ON]を押し、電源を切るには、[■][OFF]を押します。(OFFは[ON]の上側のキーボードに印刷してあります。「[■][OFF]を押す」とは、シフト・キー（[■]）を押した後に[ON]を押すことを意味します。)

計算機の電源が入っているときには、[ON]は中断（ATTN、attention）キーの働きをして、コマンド行内の文字などをクリアし、実行中のプログラムを止めます。（[ON]の下側のキーボードにATTNと印刷してあります。）

13. 数値入力　数値をキー入力するには、数字キーの[0]～[9]と[CHS]（change sign、符号の変更）、[EEX]（enter exponent、指数部の入力）を使います。小数点記号（36ページ）として点を使うには、整数部と小数部を区切るために[.]を押します。数値入力は39ページで説明します。

14. 算術演算　算術機能は40ページの「単項演算関数」と41ページの「二項演算関数」で説明します。

15. 1字取り消し　[◀]を押すとキー入力した最後の文字が消えます。

16. メニュー切り替え、次のメニュー行　メニュー切り替えキーを押すと、メニュー・キーにコマンドが割り当てられます。例えば、[TRIG]を押すとTRIGメニューに切り変わります。違うメニューに切り替えるには、目的のメニュー切り替えキーを押します。

メニュー記号が表示部に出ていないときには、カーソル機能が働いています。カーソル機能（[INS]～[▶]）はメニュー・キーの上側に白で印刷してあります。メニュー記号が出ているときに、[◀▶]を押すとカーソル機能に切り替わります。以前のメニューに戻るには、もう一度[◀▶]を押します。

各メニューにはメニューが2行以上あって、各行に6個以内のコマンドが出ます。[NEXT]を押すとそのメニューの次の行に切り替わります。[■][PREV]を押すとその前の行に切り替わります。

左側のキーボードにも別のメニュー切り替えキーがあります(2参照)。全メニューをABC順に並べて、各メニュー内のコマンドに簡単な説明を付けたものが、付録Dの「メニュー案内」にあります。

17. メニュー・キー　このキーの働きは表示部のメニュー記号で決まります。メニュー記号が出ていないときには、各キーの上側に白で印刷してあるカーソル機能になります。

18. メニュー記号　各メニュー・キーのその時点の働きを表しています。

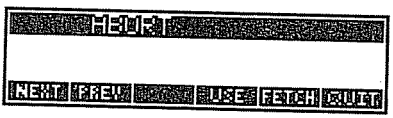
19. スタックの段　その時点で処理中のオブジェクトがスタックに見えます。番号が付いた各段（1段目や2段目、以下同様）ごとに1個のオブジェクトが入ります。

コマンドのカタログ機能

HP−28Sには全コマンドのカタログが入っています。どのコマンドもカタログでその使用法、つまりそのコマンドに必要な引き数（独立変数など）が分かります。カタログ内の全コマンドの一覧は、Reference Manual（コマンド解説書）の末尾の"Operation Index"（操作索引）にあります。

カタログを開始してみましょう。

[CATALOG]



最初のコマンドはABORTです。
カタログ表示になっている間は普通の計算機操作は停止しています。
NEXT または PREV を押すと別のコマンドのカタログ表示に変わります。USE を押すとそのコマンドに必要な引き数が見られます。FETCH または QUIT を押すとカタログ表示が終了して、普通の計算機操作に戻ります。

NEXT と PREV を押してカタログ内を動いてみてください。このキーを押したままにすると連続して移動することができます。

文字キーを押すと、その文字で始まる最初のカタログ内容に変わります。Tで試してみましょう。

T

Tの文字で始まる最初のコマンドはTAN関数です。左側のキーボードにあるアルファベットでない記号キーを押すと、その記号で始まる最初のカタログ内容に変わります。Σで試してみましょう。

▒ Σ

Σで始まる最初のコマンドはΣ＋コマンドです。左側のキーボードの記号キーを押したときに、その記号で始まるコマンドがないと、アルファベット以外で始まる最初のコマンドの＋に変わります。

#

＋の使用法を調べてみましょう。

NEXT USE

これは2個の実数の足し算ができることを表しています。次の組み合わせを調べてみましょう。

NEXT

これは1個の実数と複素数の足し算ができることを表しています。次の組み合わせを調べてみましょう。

NEXT

これは実数と複素数はどちらが先でもよいことを表しています。

残り14種の組み合わせも調べてください。最後の組み合わせは次のようになります。

組み合わせの点検が終わったら、主カタログに戻りましょう。

これで別のカタログ内容に移動してその引き数の組み合わせを調べることができます。カタログで調べることが終わったら、普通の計算機操作に戻りましょう。

別な方法として、 を押してカタログから抜け出すことができ、こうするとコマンド行にその時点のコマンド名を自動的に書き込みます。

第2章

直接計算の実行

HP−28Sで計算する方法は2種類あります。前章でやったように、スタックを使って計算する方法と、計算を表す数式を入力して計算することもできます。前章でやった計算は (15+23)×sin30°でした。

ここでは数式を使ってどう計算するかを説明します。

スタックをクリアしてTRIGメニューに切り替えます。

▒ CLEAR TRIG

数式を始めましょう。

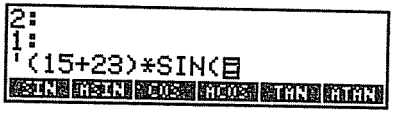
'

カーソルの形が変わって、数式入力状態を表しています。数式をキー入力するにつれてこの入力状態の働きが分かっていただけます。
数式の最初の部分をキー入力しましょう。

(15 + 23 ▒)

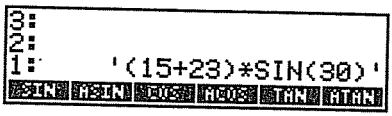
数式入力状態なので、[+]を押すとコマンド行に文字＋を書き込むだけで、コマンドは実行されません。
数式を続けましょう。

[×] [SIN]



数式入力状態なので、[×]を押すとコマンド行に＊に書き込み、[SIN]を押すとコマンド行にSIN（を書き込んで、コマンドは実行されません。
数式を完成させてスタックに入れましょう。

30 [ENTER]



閉じ括弧 ）と閉じ記号の ' が自動的に追加されています。
数式を評価してみましょう。

[EVAL]

数式がスタックから消えて、結果の19が１段目に返りました。

これで計算が終わりました。

$$(15+23)\times\sin 30^\circ=19$$

この計算をするために、教科書にあるような数式を書き込みましたが、こちらの方が数式をキー入力して評価するのが簡単でした。逆に、計算の中間結果を見たり、考えながら計算を続けるには、スタックで計算する方が簡単です。結果はどちらでも同じです。

スタック計算と数式との関係は58ページからの第４章「計算の繰り返し」でも取り上げます。この章では数値の代わりに名称を使って、スタックで計算して数式にまとめます。

数値の入力と表示

数値の表示方法に影響があるモードがいくつかあります。どう変わるかを見るために、スタックに 2 / 3 という数値を入れることにします。
スタックに 2 を入れましょう。

2 [ENTER]

3 で割りましょう。

3 [÷]

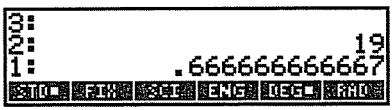
結果の 2 / 3 が 1 段目に返りました。この結果は、初期設定値の小数点記号と数値表示形式で表示したものです。これからこれを変えてみることにします。

小数点記号の変更

アメリカと日本では数値の整数部分と小数部分を区切るために点を使っています。この点の役割りから小数点と呼びますが、この数値区切りを基点記号 (radix mark) と呼ぶこともあります。

ヨーロッパを含む大部分の国では基点記号としてコンマを使っています。コンマに切り替えるには次のようにします。
MODEメニューに切り替えましょう。

[MODE]



MODEメニューの最初の行になりました。MODEメニューの次の行に切り替えます。

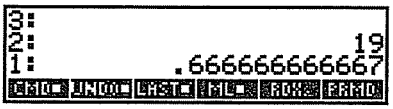
NEXT

基点記号としてコンマに切り替えましょう。

RDX.

小数点が基点記号のコンマに変わり、メニュー記号 RDX. にRDX,モードになっていることを表す四角も見えます。
RDX,モードを取り消して小数点に戻りましょう。

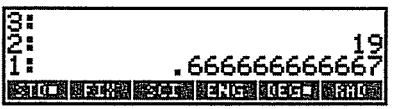
RDX.



数値表示形式の選択

小数部分の表示桁数を指定することができます。
MODEメニューの最初の行に戻りましょう。

NEXT



NEXT を押すことでメニューの最後の行からは最初の行に戻ることができます。MODEメニューは2行だけなので、 NEXT を押すことで1行目に戻りました。

このメニューで数値表示形式の4種が選べます。これはSTD (standard、標準) とFIX (fixed、小数部分固定) SCI (scientific、一般科学向け指数部付き)、ENG (engineering、工学式指数部付き) の4種です。この時点で

メニュー記号のSTDの部分に四角があるのは、STDがこの時点の選択であることを表します。STD状態では数値によって小数部分の桁数が変わります。例えば、整数では小数部分を表示しません。上例の表示では小数部分が最大の12桁まで見えています。

残り 3 種の表示形式では、表示すべき数値に関係なく、指定した小数部分の桁数（0 ～11）で表示します。これから小数部分を 2 桁に指定して全部の表示形式を見てみましょう。実際には表示する数値を四捨五入しているだけで、計算機内部の数値は変化しません。

2 / 3 を小数点 2 桁に四捨五入した表示を見ましょう。

2 FIX

2 / 3 の仮数部（ここでは小数部分を 2 桁に四捨五入したもの）と指数部の表示を見ましょう。

2 SCI

Eの前を仮数部、Eの後を指数部と呼びます。数値は仮数部に10の指数部乗を掛けたものです。仮数部は必ず 1 ～9.99999999999 の範囲内です。

2 / 3 の仮数部（ここでは小数部分を 2 桁に四捨五入したもの）と工学的指数部（これは自動的に 3 の整数倍になります）の表示を見ましょう。

2 ENG

指数部を 3 の整数倍にしたために、仮数部の小数点位置が変わったり、仮数部の末尾に 0 が付くことがあります。
標準の数値表示形式に戻しておきましょう。

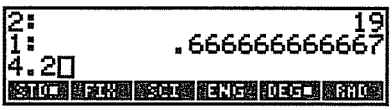
STD

数値のキー入力

数値を仮数部と指数部に分けてキー入力することもでき、数値は仮数部に10の指数部乗を掛けたものになります。仮数部と指数部のどちらかまたは両方を負数にすることもできます。

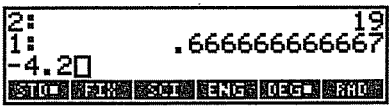
例として、-4.2×10^{-12}という数値をキー入力しましょう。
まず仮数部の数字をキー入力します。

4.2



もし間違えたら、◀を押して間違いを取り消してから正しい数字をキー入力します。
次に仮数部を負数にします。

CHS

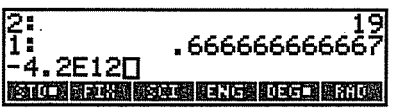


CHSはchange sign（符号の変更）を略したもので、CHSをもう1回押すと仮数部が正数になってしまいます。
次に指数部を始めましょう。

EEX

EEXはenter exponent（指数部の入力）を略したものです。コマンド行に数値の指数部を表すEが入りました。指数部がない数値に間違えてEEXを押してしまったら、間違えた数字を取り消したように、◀を押してEを取り消すことができます。
指数部の数字をキー入力しましょう。

12



指数部を負数に変えましょう。

CHS

数値をスタックに入れましょう。

ENTER

負数のキー入力に CHS を使うのを忘れないでください。例えば、この説明書のキー操作が－ 4 × になっていたら、 4 CHS × と押してください。

単項演算関数

1 個の数値を処理する関数（例えば、数値の符号を変えたり、平方根を求める）を単項演算関数と呼びます。どの単項演算関数もスタックの 1 段目に入っている数値を処理します。キーボードに直接ある単項演算関数は次の 4 種です。

- CHS を押すと数値の正負が反転します。
- ▒ 1/x を押すと数値の逆数が返ります。
- ▒ √x を押すと数値の平方根が返ります。
- ▒ x² を押すと数値の二乗が返ります。

数値をキー入力したばかりのときには、単項演算関数を実行する（つまり、上記のキーを押す）前に ENTER を押す必要はありません。関数キーを押すと自動的にENTERを実行してから目的の関数を実行するからです。例えば、1 / 8 は次のように計算できます。

8▒ 1/x

二項演算関数

足し算のように、2個の数値を処理する関数を二項演算関数と呼びます。どの二項演算関数もスタックの1段目と2段目の数値を処理します。

36ページの2を3で割ったときのように、関数用の2個の引き数をキー入力するときには、2個の引き数の間を [ENTER] で区切ることが必要です。(関数が処理する対象を引き数と呼びます。)前の計算の結果としてスタック1または2個の引き数が入っているときには、[ENTER] を押す必要はありません。

足し算と引き算

36+17を計算しましょう。

36 [ENTER]
17 [+]

結果は53です。
足し算では数値の順序は問題ではありません。しかし、引き算では数値の順序が重要です。つぎに91−27を計算しましょう。

91 [ENTER]
27 [−]

結果は64です。

掛け算と割り算

13× 6 を計算しましょう。

13 [ENTER]
6 [×]

結果は78です。

掛け算では数値の順序は問題ではありません。しかし、割り算では数値の順序が重要です。次に182/14を計算してみましょう。

182 [ENTER]
14 [÷]

累乗と累乗根

累乗（指数乗またはべき乗）と累乗根（指数乗根またはべき乗根）の計算には数値の順序が重要です。5^3を計算してみましょう。

5 [ENTER]
3 [▒] [^]

結果は125 です。
$\sqrt[4]{2401}$を計算するには、まず2401をスタックに入れます。

2401 [ENTER]

次に2401の 1 / 4 乗を計算しましょう。

4 [▒] [1/x] [▒] [^]

結果は 7 です。

パーセント

85の40%を計算しましょう。

85 [ENTER]
40 [%]

結果は34です。
パーセントでは数値の順序は問題ではありません。しかし変化率では数値の順序が重要です。60から75への変化率を計算してみましょう。

60 [ENTER]
75 [%CH]

結果は25で、これは60の25%増が75であることを意味しています。

1段目と2段目の入れ替え

数値の順序が重要な関数（引き数、割り算、累乗、累乗根、変化率）のために、[ ] [SWAP] を押して数値の順序を入れ替えることができます。例えばスタックに25があって、30−25を計算したいと仮定してみます。
30をキー入力しましょう。

30

25と30の順序を入れ替えましょう。

[ ] [SWAP]

[SWAP] を押すと自動的にENTERを先に実行します。

30から25を引きましょう。

−

結果は 5 です。

スタックからのオブジェクトのクリア

ここの例題をやるたびに、スタックに数値が増えていきます。スタックにオブジェクトを入れるたびに、スタックはどんどん大きくなって（つまり、段数が増えて）、操作でそれを使うかクリアするまで、このオブジェクトはスタックに残っています。

このオブジェクトは 1 度に 1 個ずつまたは 1 度に全部をクリア（削除）することができます。
1 段目の数値をクリアしましょう。

DROP

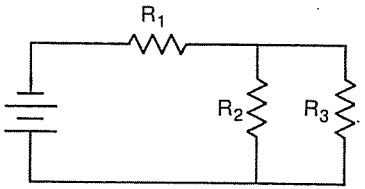
2 段目から上のオブジェクトが 1 段ずつ下に移動しました。
スタックから全オブジェクトをクリアしましょう。

▪ CLEAR

計算を開始する前にスタック全体をクリアしておくとよいでしょう。経験を積んでいくと、スタックにはその時点の問題に関連したオブジェクトだけがあって、前回のオブジェクトが残っていないほうがよいことがわかると思います。

複雑な計算

複雑な計算をするときには、スタックが中間結果を一時的に保存する役目をします。この一時的保存は自動的に働きます。例えば、次の回路の合成抵抗を計算したいものと仮定してみましょう。



この回路の合成抵抗の公式は次のようになります。

$$R_{total}=R_1+\frac{1}{\frac{1}{R_2}+\frac{1}{R_3}}$$

R_1、R_2、R_3の抵抗値がそれぞれ 8 、 6 、 3 オームだとすると、次のように計算します。

$$R_{total}=8+\frac{1}{\frac{1}{6}+\frac{1}{3}}$$

次のように計算を進めます。
スタックに 8 を入れましょう。

8 [ENTER]

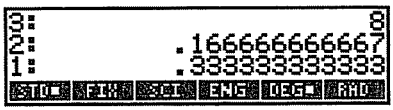
この 8 に残りの計算部分を足すまで、この 8 をスタックに残しておくことにします。

1 / 6 をスタックに入れましょう。

6 [1/x]

1 / 3 もスタックに入れましょう。

3 [1/x]



6 と 3 の逆数を足しましょう。

[+]

合計の逆数を求めましょう。

[1/x]

R_{total}の計算を終わらせましょう。

[+]

結果は10オームです。

関数を間違えて実行したとき

HP-28Sには関数を間違えて実行したときに逆戻りできるような回復機能があります。次の手順で１個の関数（単項演算関数と二項演算関数のどちらでも）の働きが逆戻りできます。

1. ▒ UNDO を押して以前のスタックの内容に戻します。
2. 間違えたときにコマンド行内に数値が入っていたら、▒ COMMAND を押してコマンド行の以前の内容に戻します。
3. 計算を続けます。

第3章

変数の使用法

変数を使うとオブジェクトを名称で参照する（つまり、オブジェクトそのものの代わりに名称で指定する）ことができるようになります。名称オブジェクトと本体のオブジェクトを結合して変数を作ります。名称オブジェクトで変数名を定義し、本体のオブジェクトで変数の内容を定義します。こうすると変数名で変数の内容を参照することができます。

変数はユーザ・メモリと呼ぶ計算機メモリの一部に記憶されます。これはスタックとは別の種類のメモリです。スタックは計算に使う一時的な数値のようなオブジェクトの記憶用に用意したものです。ユーザ・メモリは繰り返して使う数値などを入れた変数の長期記憶用に用意したものです。

この章ではまず数値変数を作りますが、これにはすぐ慣れると思いますが、その後で作るプログラム変数はなじみがないかも知れません。HP-28S内部では、プログラムに特には区別用の名称がなく、単に別なオブジェクト型であるだけです。プログラムを変数に記憶させるときに、数値のときと同様に、名称を付けます。これで変数名でプログラムを実行することができます。

変数の作成、呼び出し、評価、変更、改名、削除の各方法はどの変数にも共通で、その内容には無関係です。この操作の均一性のおかげでHP-28Sは使用法が簡単なのに強力な計算機になっています。例外が少ないのと柔軟性に富んでいるからと言えます。

変数の基本操作

一番簡単な変数は数値変数です。ここでは数値変数の作成、呼び出し、評価の方法を説明します。

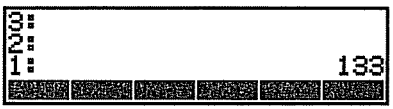
数値変数の作成

計算で133という体積値を繰り返して使うものと仮定します。次のようにVOL（体積、volumeの略）という名称の変数を作りましょう。
スタックをクリアしてUSERメニューに切り替えます。

[ON] [CLEAR] [USER]

USERメニューで変数名が見えます。ここが空白なのはまだ変数を作っていないからです。
1段目に数値を入れましょう。

133 [ENTER]

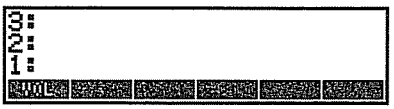


1段目に名称の ' VOL ' を入れましょう。

[']VOL [ENTER]

右側の ' は自動的に追加されています。数値の133は2段目に上がりました。
（実際にはここで [ENTER] を押す必要はないのですが、はっきりと見てもらうために押してみました。）
変数VOLを作りましょう。

[STO]



数値と名称がスタックから取り出されて、名称がVOLで内容が133の変数になりました。これでUSERメニュー内にVOLが見えるようになりました。

数値変数の呼び出し

次に作ったばかりの変数VOLの値をスタックに返してみましょう。
USERメニューの利点を使って、スタックに名称VOLを入れましょう。

' VOL ENTER

VOLの内容を呼び出しましょう。

RCL

これがVOL内に記憶させた数値です。

数値レジスタのある計算機の操作に慣れていれば、呼び出し操作についてはなじみがあると思います。HP-28Sでは、変数の内容そのものを呼び出すことはそれほど多くなく、変数を評価することの方がかなり多くなります。

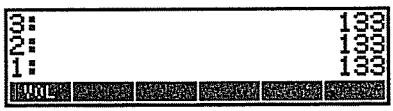
数値変数の評価

数値変数では、偶然ですが「評価」には「呼び出し」と同じ効果があって、数値変数を評価するとスタックに数値が返ります。
評価の方が操作が簡単です。(55ページでプログラムを作ると、評価と呼び出しの働きが違うことがわかります。)
評価してスタックにVOLの値を返しましょう。

VOL

引用符を付けずに名称だけを入力してもVOLを評価することができます。

VOL ENTER



変数の値の変更

変数を作ったときと同じ手順を使って、変数の値を変えることもできます。古い値を新しい値に置き替えてしまいます。

ここでVOLの値を151に変えてみましょう。
コマンド行に新しい値を書き込みます。

151

カーソルの形は四角い枠です。次の操作でカーソルの形が変わります。
コマンド行に変数名を書き込みましょう。

[']　[VOL]

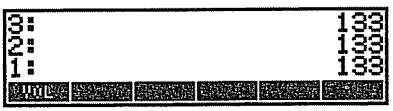
[']を押したときにカーソルの形が変わって、入力モード(キーを押したときに計算機がどう応答するか) が新しくなったことを示します。

最初はコマンド行が普通入力モードで、キーボード計算に使います。[']を押すと、名称または数式の始まりを表し、コマンド行が数式入力モードに変わり、名称や数式の入力に使えるように機能が変化します。

- [+]を押すと文字+を書いたように、関数キーを押してもそのコマンドを実行しません。
- USERメニューのメニュー・キーを押すと、その変数名を書き込むだけで、変数は評価しません。

新しい値を変数に記憶させましょう。

[STO]



新しい値を調べましょう。

VOL

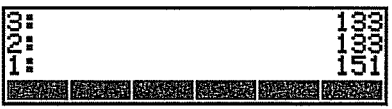
変数の削除

変数VOLの使用が終わったら、これをユーザ・メモリから削除しましょう。コマンド行に変数名を書き込みましょう。

' VOL

(引用符の ' は変数の評価を防ぐために必要です。)
ユーザ・メモリから変数VOLを削除しましょう。

PURGE



USERメニューからVOLの記号が消えました。

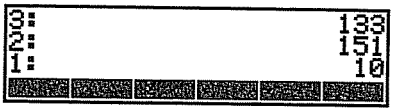
変数名の変更

同じ値の新しい変数を作って、元の変数を削除する方法で、変数名を実質的に変えることができます。

この部分ではまず変数を改名するのに必要な手順を実行し、次に同じ手順を入れたプログラムを書き込み、最後にこのプログラムを変数に記憶させてから名称で実行してみましょう。

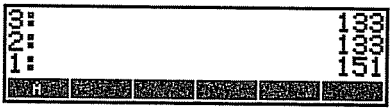
準備として、後で改名するための何かが入っている変数（例えば、値が10の変数A）を作りましょう。
値の10をスタックに入れます。

10 [ENTER]



変数Aを作りましょう。

[']A [STO]

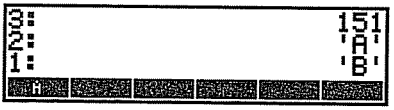


USERメニューにAが見えるようになりました。
AからBに改名したいものと仮定します。古い名称をスタックに入れましょう。

['] [A] [ENTER]

新しい名称をスタックに入れましょう。

[']B [ENTER]



これで準備が完了です。ユーザメモリに変数があり、古い名称がスタックにあり、新しい名称もスタックにあります。この古い名称と新しい名称がプログラムの引き数で、プログラムは引き数がスタックにこの順序で入っていることを前提にしています。これ以後の手順がプログラムになります。

手順の中には汎用のスタック操作用コマンドが3個(OVERとROT、SAVE)あります。これを実行するときにその働きも見ることになります。
古い名称を1段目にコピーしましょう。(STACKメニュー内のOVERコマンドを使います。)

[▒] [STACK] [OVER]

変数の内容を呼び出しましょう。

RCL

古い名称を1段目に移しましょう。(回転を意味するROTコマンドを使います。)

ROT

古い変数を削除しましょう。(新しい変数を作る前に古い変数を削除することで、同じ値の別なコピーを作ることを防止します。)

PURGE

正しい順序になるように内容と新しい名称を入れ替えます。

SWAP

新しい変数を作りましょう。

STO

これで以上の手順を自動化するプログラムを作ることができます。

プログラム変数の作成

まずプログラムをキー入力し、次にこれを変数内に記憶させましょう。

プログラムの識別記号でプログラムを開始します。

«

カーソルの形が変わって状態表示の α が見えるようになり、どちらも文字入力モードを表します。プログラム可能な操作用のキーを押すとその操作名が文字になってコマンド行に書き込まれます。プログラム不能な操作（例えば、文字を取り消すための ◀ を押すことなど）は機能を実行します。
前に実行した手順をキー入力しましょう。

OVER RCL
ROT PURGE
SWAP STO
ENTER

右側の識別記号 » は自動的に追加されています。
このプログラムを変数RENAME内に記憶させましょう。

' RENAME STO

USERメニューを調べてみましょう。

USER

USERメニュー内にRENAME（実際には始めの4文字RENA）が見えます。

これでRENAMEが実行できます。最初はRCLを使った遠回りな方法、次にUSERメニューを使った普通の方法で実行してみましょう。こうすることによりプログラム変数の呼び出しと評価の違いが明らかになります。

プログラム変数の呼び出し

この例では、変数BをCに改名しましょう。
スタックに古い名称と新しい名称を入れます。

' B ENTER
' C ENTER

プログラムRENAMEを呼び出しましょう。

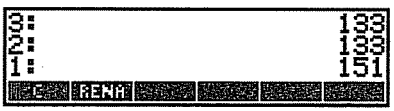
' RENA ▒ RCL

どんな変数でも、RCLでは単に変数の内容がスタックに返るだけです。

プログラム変数の評価

スタックにあるプログラムを実行するには、わざわざ評価という操作をしなければなりません。

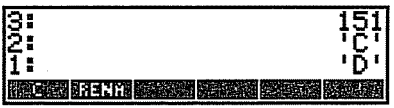
EVAL



BからCに改名したことがUSERメニューでわかります。

実際には実行するためにスタックにプログラムを呼び出すことは不要ですが、プログラムに対してRCLがどう働くかとEVALでプログラムがどう実行されるかを見るためにこうしてみました。次にプログラムを実行する簡単な方法を見ることにしましょう。
今度はCをDに改名します。古い名称と新しい名称をスタックに入れましょう。

' C ENTER
' D ENTER



CをDに改名しましょう。

RENAME

```
3:                    133
2:                    133
1:                    151
 D   RENAME
```

USERメニューでCがDに改名されたことがわかります。USERメニューのキーを1個押すだけでプログラムを簡単に実行できました。

引用符で囲んだ名称と囲まない名称

上例で、引用符付きと引用符なしの2種の方法で変数名を使いました。引用符の ' は重要で、これで変数の名称と変数の内容を区別します。引用符付きと引用符なしの名称を使う目的の要点は次の通りです。

- 引用符付きの名称を使うとその名称そのものを表します。引用符で名称の評価を防止するので、名称がスタックに入ってコマンドの引き数になります。この章ではSTO、RCL、PURGEとプログラムRENAMEの引き数として、引用符付きの名称を使いました。
- 引用符なしの名称を使うとその名称が付いた変数が評価されます。引用符なしの名称はスタックに入らず、その代わりに、その変数内に記憶しているオブジェクトをその型に応じて処理するので、数値変数はスタックに数値を返し、プログラム変数はプログラムを実行します。これ以外のオブジェクト型ではどうなるかについては、この説明書の後の方に出てきます。

変数に対応していない名称に引用符を付けないでキー入力すると、引用符付きの名称がスタックに入ります。

第4章

計算を繰り返す方法

この章では数値変数が混っている数式を作り、次にSOLVE という機能を使って数式内の数値変数のさまざまな値に対応させて数式の評価をしてみます。第2章では数値が入っている数式をキー入力して、その数式を評価しました。この章では記号表現をした引き数として名称を使い、スタック上で記号どうしの計算をして数式を作ります。その後でSOLVE を使って変数に数値を代入したり数式の評価をします。評価をするたびに、変数内のその時点の値での計算が行われます。いくつかの変数の値を変えても、その数式を評価する操作だけで、新しい値で再計算できます。
第3章では数値変数とプログラム変数を作りました。この章では数式変数と名称変数を作ります。(どんなオブジェクトでも変数に記憶させることができます。) それに変数の管理用のディレクトリについても学びます。

数式の作成

第2章の「複雑な計算」でやった抵抗計算を繰り返しますが、今回は引き数として数値でなく名称を使います。抵抗計算の公式は次のものでした。

$$R_{total}=R_1+\frac{1}{\frac{1}{R_2}+\frac{1}{R_3}}$$

スタックをクリアし、メニュー・キーをカーソル機能に切り替えましょう。

▒ CLEAR

```
4:
3:
2:
1:
```

メニューが見えていたら、◀▶を押してカーソル機能に切り替えてください。

名称'R1'をスタックに入れましょう。

' R1 ENTER

```
4:
3:
2:
1:                              'R1'
```

右側の'が自動的に追加されました。R1は計算の残り部分を加えるまでスタックに残しておきます。

R2の逆数をスタックに入れます。

' R2 ENTER ▒ 1/x

```
4:
3:
2:                              'R1'
1:                         'INV(R2)'
```

R3の逆数もスタックに入れます。

' R3 ENTER ▒ 1/x

```
4:
3:                              'R1'
2:                         'INV(R2)'
1:                         'INV(R3)'
```

R2とR3の逆数を足しましょう。

+

```
4:
3:
2:                              'R1'
1:                 'INV(R2)+INV(R3)'
```

合計の逆数を求めましょう。

▒ 1/x

```
3:
2:                              'R1'
1:  'INV(INV(R2)+INV(R3)
    )'
```

R1とこの逆数を足しましょう。

⊞

```
3:
2:
1:  'R1+INV(INV(R2)+INV(
    R3))'
```

これがR_{total}を表す数式です。
必要な括弧に気を付ければ、この数式全体を直接キー入力することもできます。どの数式もスタックで計算したのと同じなので、操作が楽なほうを選んでください。
この章の後の方でこの数式を変数に記憶させますが、その前にディレクトリを作ってこの章の例題をひとつのグループにまとめましょう。

ディレクトリの作成

ディレクトリとは変数をグループにして管理するためのものです。これまではメモリのリセット後でも失われない組み込みのHOMEディレクトリで作業していました。この章ではHOMEディレクトリ内に子ディレクトリを作り、その子ディレクトリ内にまた子ディレクトリ（HOMEから見ると孫ディレクトリ）を作ります。

この章で使うディレクトリについての概念は次の通りです。

- 1度に1個のディレクトリだけが使用できます。そのディレクトリ内の変数だけがUSERメニューに現れます。
- ディレクトリAの内部にディレクトリBがあると、AをBの親ディレクトリと呼び、BをAの子ディレクトリと呼びます。
- その時点のディレクトリを起点にしてその親ディレクトリを探し、次にその親ディレクトリを探すことを続けると必ずHOMEディレクトリに戻ります。このディレクトリ名の順序を逆に並べたものをディレクトリの経路（Path）と呼びます。

PATHコマンドを実行すると、その時点の経路を調べることができます。

MEMORYメニューに切り替えましょう。

▒ MEMORY

```
2:
1:  'R1+INV(INV(R2)+INV(
    R3))'
MEM MENU ORDER PATH HOME CRDIR
```

経路を調べてみましょう。

PATH

PATHが返すリストは必ずHOMEから始まってその時点のディレクトリで終わります。HOMEは全経路の起点であり別のディレクトリをひとつも作ってなければ、HOMEがその時点のディレクトリとなります。
電気についての計算作業だけをひとつにまとめるために、EEという名の子ディレクトリを作りましょう。

' EE CRDIR

EEディレクトリに切り替えましょう。

EE ENTER

また経路を調べてみましょう。

PATH

この時点のディレクトリはEEです。EEディレクトリに切り替えた効果を見るために、USERメニューを見ましょう。

USER

前章で作ったRENAMEプログラムは見えません。USERメニューにはこのディレクトリ（EE）内の変数だけが見えRENAMEはHOMEディレクトリ内にあるので見えません。
しかし、この時点の経路（HOME EE）にあるディレクトリ内の変数は経路を自動的にたどることによって名称で見つけることができるので、名称を入力することでRENAMEの実行ができます。

これがHP-28Sのディレクトリの利点の一つで、RENAMEのような一般的なプログラムはHOMEディレクトリに入れておくと、いつでも実行できて、しかもUSERメニューが雑然となることはありません。

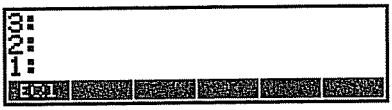
これから新しいディレクトリEEで作業することができます。

スタックから経路のリストを2個削除しましょう。

DROP　DROP

この数式をEQ1（equation 1、数式1）という名称の変数に記憶させましょう。この名称を付けたのは後で別の数式を作るからです。

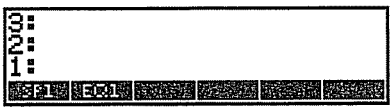
' EQ1 STO



USERメニューに変数名EQ1 が見えます。
この数式でさまざまなケースの問題を解くようになると、それぞれを別々に扱えれば便利です。そのためには、問題別の子ディレクトリを作ってやり、そこに各ケースの値を別々に入れることができます。

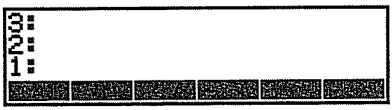
最初の問題用に子ディレクトリSP1(series-parallel 1、直並列1)を作りましょう。

' SP1 ENTER
CRDIR ENTER



USERメニューに新しい子ディレクトリの名称が見えます。このメニュー・キーを押してSP1 に切り替えましょう。

SP1



この時点のディレクトリ（SP1 ）が空なので、USERメニューはまだ空です。

この時点の経路を調べてみましょう。

PATH [ENTER]

HOMEまたはEEディレクトリ内にある変数はここから名称で探すことができます。これはこのディレクトリがこの時点の経路上（HOME EE SP1）にあるからですが、USERメニューに見えるのはこの時点のディレクトリ（SP1）内にある変数名だけです。

これで数式EQ1 をSOLVE で使う準備ができました。

計算の繰り返しにSOLVE機能を使う

SOLVE を使って数式を解くには次の基本手順を使います。

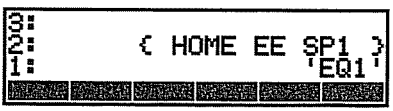
1. 数式（または数式の名称）をEQ（equation、数式）という名の変数に記憶させます。SOLVE はいつでもこの変数名を使うようになっています。
2. SOLVE メニューを使って変数に値を代入します。
3. SOLVE メニューを使って数式を評価します。
4. 別の値で計算したいときは手順2と3を繰り返します。

これからの例題で手順を説明します。

手順1　変数EQ内に名称EQ1 を保存します。

この手順で、なぜ変数に名称を記憶させるのか？　なぜEQ内に数式そのものを記憶させないのか？　と驚いたのではないでしょうか。単純な理由は数式全体よりも名称の方が短くて覚えやすいからです。それに、後でこの数式を呼び出したり、別の数式に切り替えるのに、この方が便利だと分かります。スタックに名称EQ1 を入れましょう。

['] EQ1 [ENTER]



引用符の ['] を入れ忘れると、スタックに数式そのものが入ります。この場合は [DROP] を押して数式を削除してから、やり直してください。

SOLVEメニューに切り替えましょう。

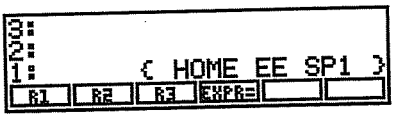
SOLV

STEQ(store equation、数式の保存)を使って変数EQ内に名称EQ1 を保存しましょう。

STEQ

手順2　変数に値を代入します。
SOLVRメニューに切り替えましょう。

SOLVR



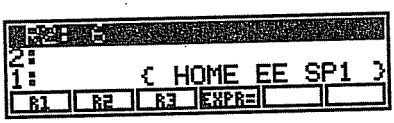
この数式内の変数名がSOLVR メニュー内に現れます。(数式に7個以上の変数があるときには、 NEXT を押すと別の変数の行が現れます。)

このメニューが普通のメニューとは違って表示されるのは、働きが違うからです。このSOLVR メニューでは変数の評価ではなく、変数への値の保存に使われます。
これで変数R1、R2、R3に値の代入ができます。まず変数R1に数値8を保存します。

8 R1

R1 を押すのはスタックに'R1'を入れて STO を押すのと同じです。
最上行に変数名とその値が白抜きで表示されます。
変数R2に数値6を保存しましょう。

6 R2



変数R3に数値3を保存しましょう。

3 [R3]

手順3　数式を評価します。

メニュー記号の[EXPR=]は「数式の値は」の意味で、これを押すと数式が評価されます。

[EXPR=]

数値(10)が1段目に返り、最上行に白抜きの文字でも現れます。

手順4　別の値で計算するには手順2と3を繰り返します。例として、R3が12のときの値を求めます。

変数R3に数値12を保存しましょう。他の値はそのまま保存されているので再入力の必要はありません。

12 [R3]

この時点の変数値で数式を評価しましょう。

[EXPR=]

新しい数値(12)が1段目に返り、最上行にも白抜きで現れます。

異なる値の組み合わせのときの使用法

R1、R2、R3の値が全部違う問題を解いた後に、前回に代入した値を使う問題に戻りたいと仮定します。問題を切り替えるごとに全部の値を入れ直すこともできますが、ここでは別の便利な方法を紹介します。これは次の3手順です。

1．別の値用の新しいディレクトリを作ります。
2．同じ計算式をEQに定義します。
3．前回のSOLVE と同様に値を代入して数式を評価します。

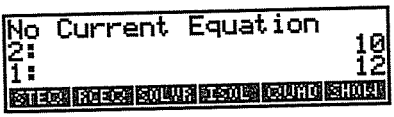
この方法を使うと次のようにディレクトリの別な利点が利用できます。あるディレクトリ内のひとつの変数にはひとつだけ値を入れておくことができますが、同じ名称の変数で異なる値を持つディレクトリはいくつでも作ることができます。

手順1　新しいディレクトリを作ります。

SP1の代わりの新しいディレクトリとして、同じ親ディレクトリ(つまりEE)内にSP2 と呼ぶディレクトリを作ります。これがディレクトリ構造の最初の分岐で、同じ親ディレクトリ(EE)内に2個の子ディレクトリ(SP1とSP2)を持つことになります。

EE内に子ディレクトリを作るために、ディレクトリをEEに切り替えます。(このまま子ディレクトリを作るとSP1 内に作ることになってしまいます。)EEディレクトリに切り替えましょう。

EE [ENTER]



警告音が鳴り、No Current Equation の表示が出て、SOLVEメニューに戻りました。これはEEディレクトリ内の'EQ'に計算式が入っていないからです。

ディレクトリSP2 を作りましょう。

' SP2
[MEMORY] CRDIR

SP2ディレクトリに切り替えましょう。

SP2 [ENTER]

この時点の経路を調べてみましょう。

PATH

ディレクトリがSP1であったときのように、この時点の経路にもHOMEとEEがありますが、今度はSP1がありません。結果として、RENAMEのようなHOME内の変数と、EQ1のようなEE内の変数は使えますが、SP1内の変数(EQ、R1、R2、R3)は使えないので、これから新しい変数R1、R2、R3を作ることになります。

手順2　同じ数式をEQに定義します。
前回と同様に、STEQを使って変数EQ内に名称EQ1を保存します。

' EQ1 [SOLV] STEQ

手順3　前回と同様にSOLVR を使って値を代入して数式を評価します。今回の例題では値を次のように仮定します。

R1＝11、　　R2＝21、　　R3＝ 7

SOLVR メニューに切り替えましょう。

SOLVR

値を代入します。

11 [R1]
21 [R2]
7 [R3]

数式を評価しましょう。

EXPR=

以前の問題に戻るためには、(EEディレクトリに戻るために)EEを実行し、(SP1ディレクトリに切り替えるために) SP1を実行し、(SOLVRメニューが働くように) SOLV SOLVR を押します。全変数の値がSP1 から抜け出したときと同じ値のままになっているはずです。

別の数式を使う方法

これまで数式EQ1 用に 2 組のデータを使いました。今度はどちらのデータでも使うことができる第 2 の数式EQ2 を作ってみましょう。基本手順は次の通りです。

1. EEディレクトリに切り替えて、新しい数式を作り、この新しい数式を変数EQ2に保存します。
2. SP1またはSP2ディレクトリに切り替えて、EQの定義を'EQ1'から'EQ2'に変更してからSOLVRを使って数式を評価します。

手順1　EEディレクトリに切り替えて、新しい数式を作り、この新しい数式を変数EQ2 に保存します。

EEディレクトリに切り替えましょう。

EE ENTER

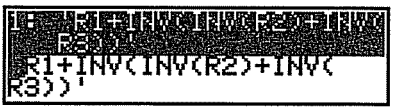
新しい数式を作りましょう。この例では、数式EQ1のコピーを修正してEQ2にします。

EQ1内に保存している数式をスタックにコピーしましょう。

USER　EQ1

数式をコマンド行にコピーしましょう。

EDIT



スタック1段目の数式が白抜き文字になったのは、コマンドの内容によって書き替わることを知らせるためです。案内表示のαが現れて、文字入力状態になっていることを示します。
これからこの数式を次の公式を表すように修正します。

$$R_{total}=R1+\frac{1}{\frac{1}{R2}+\frac{1}{R3}+\frac{1}{R3}}$$

カーソルをコマンド行の下側の行に動かしましょう。(カーソル機能はメニュー・キーの上側に白で印刷してあります。)

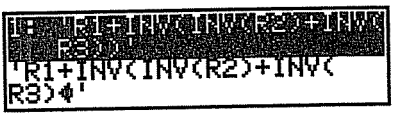
▼

カーソル機能が使えるのはコマンド行が画面にあって、メニュー記号が出ていないときです。◀▶を押すことでカーソル機能に入ったり出たりすることができます。EDITを押すと自動的にカーソル機能に入ります。
カーソルをR3の項の直後に動かしましょう。

▶ ▶ ▶

挿入モードに入りましょう。

INS



カーソルの形が矢印に変わって、文を入力するとカーソル位置の文字の左側に挿入されます。(INS をもう一度押すと重ね書きモードに戻って、カーソル位置の文字を書き替えます。)
R3の2番目の項をキー入力しましょう。

+ ▒ 1/x (R3

スタック1段目の数式をコマンド行内で修正した数式に置き替えましょう。

ENTER

新しい数式を変数EQ2 内に保存しましょう。

' EQ2 STO

手順2　SP1 ディレクトリまたはSP2 ディレクトリに切り替えて、EQの定義を'EQ1 'から'EQ2 'に変更し、SOLVR を使って数式を評価します。
この例では、SP1 ディレクトリ内の値を使います。
SP1 ディレクトリに切り替えましょう。

SP1

EQの定義をEQ1 からEQ2 に変えましょう。

' EQ2 SOLV STEQ

SP1 の値を使って数式EQ2 を評価しましょう。

SOLVR EXPR=

もしもSP2 の値を使ってEQ2 を評価したいときには、(EEディレクトリに戻るために) EEを実行してから、手順2のSP1 をSP2 に取り替えて繰り返してください。

HOMEディレクトリへの復帰

これで電気工学の問題はとりあえず完了したと仮定すると、HOMEディレクトリに帰ることができます。HOMEは組み込みディレクトリで、この名称はMEMORYメニュー内にあります。
HOMEディレクトリに切り替えましょう。

MEMORY HOME

USERメニューを調べてみましょう。

USER

メニュー記号の EE はこの章で作ったもの (EQ1 、EQ2 、子ディレクトリのSP1 とSP2 、その中の全部の変数) をすべて表しています。これがディレクトリの主な利点で、親ディレクトリから見ると、ひとつのディレクトリ全体 (その内部の変数とそのディレクトリ自体の子ディレクトリ) がディレクトリ名として見えるだけで済みます。

この章の要点

この章で見てきた使いこなすためのこつをまとめておきます。

- 関連した一連の問題ごとにディレクトリを作ります。
- 問題で使う数式ごとに別の変数に保存します。
- 問題の特定の値ごとに子ディレクトリを作ります。
- どのような数式や値の組み合わせにもSOLVEを使います。

第5章

実数を扱う関数

この章ではTRIGとLOGS、そしてREALメニューを紹介します。TRIGメニューには三角関数と角度を扱うためのコマンドが入っています。LOGSメニューには対数関数と指数関数、双曲線関数が入っています。REALメニューにはこれ以外の実数用コマンドが入っています。

この3種のメニューの全コマンドの要点は「付録D　メニュー案内」にあります。詳細はReference Manualの〝TRIG〟または〝LOGS〟、〝REAL〟を見てください。

三角関数

ここでは角度単位モードの切り替えと、円周率πを使った計算、角度の変換を取り上げます。

角度単位モードの選択

この計算機は角度の引き数と結果を度単位(円の1／360を1単位にする)またはラジアン単位(円の1／2πを1単位にする)のどちらでも解釈できます。初期設定は度単位になっています。ここの例題では、ラジアン単位に切り替えます。

スタックをクリアしてMODEメニューに切り替えましょう。

[CLEAR] [MODE]

右端の2個のメニュー記号の DEG (degrees、度)と RAD (radians、ラジアン)でこの時点の角度単位を表します。 DEG 記号の中に小さな四角があるのは、この時点の角度単位は度であることを示します。
ラジアン単位に切り替えましょう。

RAD

ラジアン単位の案内表示 (2π) が現れ、メニュー記号が変わりました。(この本の説明図では状態表示を省略しています (2π) の表示位置は27ページの図を見てください。)
TRIGメニューの最初の行に切り替えましょう。

TRIG

どれも単項演算関数で、スタック1段目の数値を処理します。実数では、角度単位モードによってSIN(sine、正弦)とCOS(cosine、余弦)、TAN(tangent、正接)の引き数をどう解釈し、ASIN(arc sine、逆正弦)とACOS(arc cosine、逆余弦)、ATAN(arc tangent、逆正接)の結果をどう表すかに影響します。

次のπの説明部分でSIN関数を使います。

πの使用法

円周率のπは超越数(無限小数)なので、有限小数の形で正確に表すことはできません。一般に、12桁精度の計算機は12桁近似値(3.14159265359)になり、普通の用途にはこれで十分です。

HP−28Sではπを正確に表す記号定数の π も扱えます。ラジアン単位では、SINとCOS、TAN関数で記号定数 π を認識して正確な結果になります。SINとCOS関数では π/2 も認識します。

これ以外の関数では、記号定数 π を使うと π が入っている数式を作ります。強制的に実数化すると、計算機は12桁の近似値を使います。

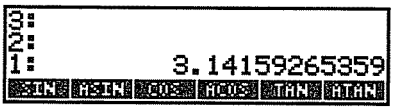
3.14159265359と π の違いをみるために、それぞれのサインを求めてみましょう。
1段目に 'π' を入れます。

[▒] [π] [ENTER]

このオブジェクトは名称のように見えますが、実際には記号定数πだけの、単項の数式です。

→NUM(to number、数値化)を使って実数の結果を求めましょう。

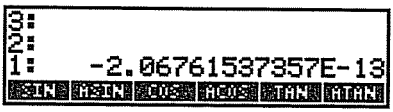
[▒] [→NUM]



πの12桁の近似値(3.14159265359)が1段目に返りました。

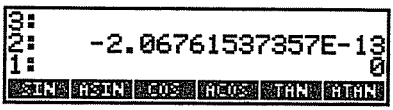
πの近似値のサインを求めましょう。

[SIN]



結果(−2.06761537357×10^{-13})が正確に0でないのは、引き数(3.14159265359)が正確にπでないからです。
今度はπのサインを求めてみましょう。

[▒] [π] [SIN]



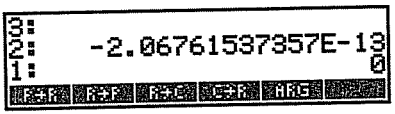
SIN 関数は記号定数πを認識するので正確な結果(0)が返りました。

角度の変換

TRIGメニューにはある測定方法による角度から別の測定方法への変換を行う角度変換コマンドも入っています。このコマンドはTRIGメニューの3行目にあります。3行目に進む前に2行目を簡単に見ることにします。

TRIGメニューの2行目に切り替えましょう。

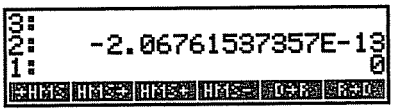
NEXT



このコマンドは複素数も扱えるので、COMPLEX メニューにも入っています。複素数は次の章で説明します。

TRIGメニューの3行目に切り替えましょう。

NEXT



ここではHMS→とD→Rコマンドを使って60進法(度、分、秒)で表した角度からラジアンで表した角度に変換します。

4種のHMS(hours-minutes-seconds、度分秒)コマンドで小数部分で分と秒を表した数値を扱う計算ができます。この数値はHMS 形式と呼ぶ次のような特別な形式でなければなりません。

h.MMSSs

このhは時(または度)を表し、MMは分を表し、SSは秒を表し、sは秒の小数部分(10進法)を表します。MMとSSはどちらも必ず2桁で表し、hとsには桁数の制限はありません。

→HMS(decimal-to-HMS、10進法から60進法へ)とHMS→(HMS-to-decimal、60進法から10進法へ)コマンドは普通の10進法形式と特別なHMS形式との間で実数を変換します。HMS+(HMS plus、HMSの足し算)とHMS−(HMS minus 、HMSの引き算)コマンドはHMS形式の数値の足し算または引き算をして、その結果もHMS形式になります。

例として、141° 26′ 15″を10進の度に変換します。
HMS形式で数値を入力しましょう。

141.2615

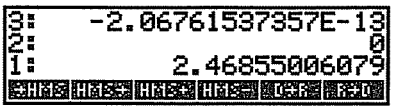
HMS形式の数値を10進の度に変換しましょう。

HMS→

このメニューの残り2種の関数は、D→R(degrees-to-radians、度からラジアンへ)とR→D(radians-to-degrees、ラジアンから度へ)で、度単位の角度とラジアン単位の角度の間で実数を変換します。

1段目の数値を度からラジアンに変換しましょう。

D→R



両方で、次の計算をしました。

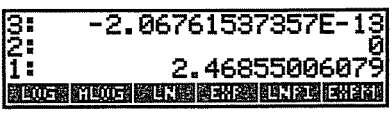
$$141° 26′ 15″ = 141.4375° = 2.46855006079 \text{ ラジアン}$$

対数関数、指数関数、双曲線関数

LOGSメニューには対数関数と指数関数(対数関数の逆関数)が入っていて、どちらにも常用と自然の2種があり、それに双曲線関数も入っています。この関数の詳細は、Reference Manualの〝LOGS〟を見てください。

LOGSメニューの1行目に切り替えましょう。

LOGS

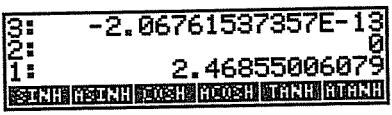


LOG(common logarithm、常用対数)とALOG(common antilogarithm、常用対数の逆関数)関数は底を10とする対数と指数関数を計算します。LN(natural logarithm、自然対数)とEXP(natural exponential、自然対数の逆関数)関数は底をeとする対数と指数関数を計算します。(eは無限小数でその近似値は2.71828182846です。)

引き数xについて、LNP1(ln plus 1、1を足した自然対数)関数は$\ln(x+1)$を計算し、EXPM(exp minus 1、指数関数引く1)関数は$(\exp\ x)-1$を計算します。引き数が0に近いときには、両方の関数とも対応する関数よりも精度が高い答えになります。(LNP1の使用例は103ページの均等払い複利計算にあります。)

LOGSメニューの2行目に切り替えましょう。

NEXT

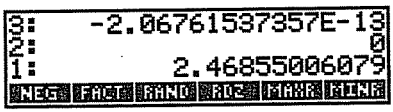


これ以外の実数関数

REALメニューには主として実数に適用する関数が入っています。

REALメニューに切り替えましょう。

REAL



NEG(negate、正負符号の反転)関数は引き数がxのときに$-x$を返します。FACT(factorial、階乗)関数は正の整数nに対してn!を返し、整数でない引き数xに対してガンマ関数の$\Gamma(x+1)$を返します。RAND(random number、乱数)関数はRDZ(randomize、無作為化)で指定した乱数の種から計算した乱数を返します。

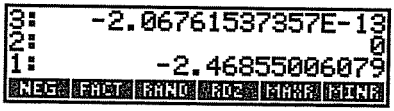
MAXR(maximum real、最大の実数)とMINR(minimum real、最小の実数)関数はHP-28Sで表現できる最大と最小の正の実数用の記号定数を返します。(記号定数を数値化するには、74ページの「πの使用法」を見てください。)

ここではNEG関数の使用法を説明します。便利なように、コマンド行がないときには CHS (change sign、正負符号の反転)を押すことでNEGを実行することができるようになっています。コマンド行にNEGを入れる(例えば、プログラムのキー入力のとき)には、 NEG または N E G を押します。

さて1段目の数値の符号を1回は CHS を押し、1回は NEG を押して、2回反転しましょう。

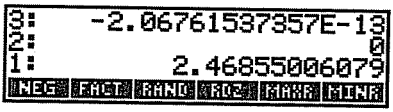
1段目の数値の符号を反転します。

CHS



もう一度反転しましょう。

NEG



新しい関数の定義

組み込み関数のように働くプログラム変数(この中に数式を使うこともできます)を作ることができます。このようなプログラム変数をユーザ定義関数と呼びますが、次の両方の条件をみたす必要があります。

- その引き数を明示することが必要です。
- 結果を正確に1個返すことが必要です。

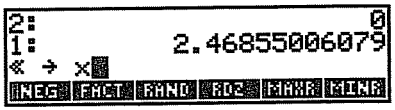
例として、cot$x$＝1／tan$x$というコタンジェント(余接)関数用の関数COTを定義してみます。

プログラムを開始しましょう。

[«]

引き数を示しましょう。

[▒] [→] [LC] X [LC]



右矢印は次の名称がローカル変数(このプログラムの内部でだけ通用する変数)を表します。

普通の変数(つまり、グローバル変数)とローカル変数を区別するために、次の約束に従うと便利です。この説明書ではローカル変数を区別するために小文字を使います。([LC]を1回押すと小文字に切り替わり、もう1回[LC]を押すと大文字に戻ります。)

関数を定義しましょう。

['] [▒] [1／x] [(] TAN [(] [LC] X

プログラムをスタックに入れましょう。

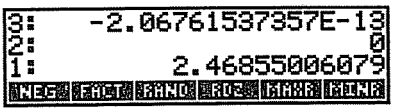
[ENTER]

閉じ括弧と右側の識別記号が自動的に追加されました。

このプログラムの意味は、スタックから(RPN方式で使用した場合)または数式から(数式中で使用した場合)1個の引き数を取り出してxと名付け、xのローカル定義を使った1／tan$x$を評価するということになります。

このプログラムを変数COTに保存しましょう。

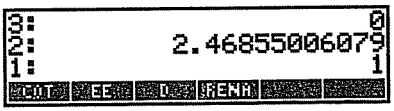
[']COT[STO]



これでHP-28Sに組み込みの三角関数のように、RPNまたは数式の中でCOTを使うことができます。

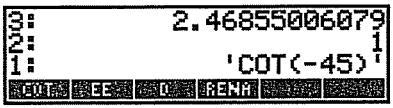
RPNでcot45°を計算してみましょう。

[MODE] [DEG]
45 [USER] [COT]



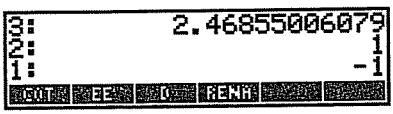
数式を使ってcot −45°を計算してみましょう。

['] [COT] [(] −45 [ENTER]



この数式を評価しましょう。

[EVAL]



第6章

複素数を扱う関数

HP-28Sは複素数を表すオブジェクト型も扱えます。例えば、複素数z＝3＋4iはオブジェクト(3、4)で表します。各複素数ごとに1個のオブジェクトになるので、複素数でも実数のときのように簡単に計算ができます。

複素数内の2個1組の実数で平面座標の1点を表すこともできます。例えば、HP-28Sは複数の複素数を使ってプロットした座標を表します。この章の84～86ページで、直交座標系と極座標系の2種の座標系を説明し、一方から他方への座標変換も取り上げます。

複素数の使用法

実数を扱う大部分の関数は同じ方法で複素数も扱えます。例えば、複素数の算術計算も実数のときと同様に、数値スタックに入れて関数を実行します。次の式を計算してみましょう。

$$((9+2i)+(-4+3i))\times(6+i)$$

スタックをクリアして9＋2iを入れましょう。

[ON] [CLEAR]
[(] 9 [,] 2 [ENTER]

$-4+3i$ を足しましょう。（-4 を入れるのには [4] [CHS] と押します。）

[(] −4 [,] 3 [+]

$6+i$ を掛けましょう。

[(] 6 [,] 1 [×]

ときによっては1個の実数の引き数から1個の複素数の結果になることがあります。
$\sqrt{-4}$ を計算しましょう。

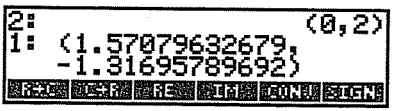
−4 [▒] [√x]

$\sin^{-1}2$ (逆正弦)を計算しましょう。

2 [TRIG] [▒ASIN]

複素数用の特別な関数はCOMPLEX メニュー内にあります。
COMPLEXメニューに切り替えましょう。

[▒] [COMPLX]



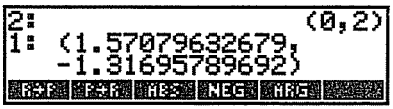
COMPLEX メニュー内の全コマンドの要約は付録Dの「メニュー案内」にあります。詳しい説明はReference Manualの〝COMPLEX〟にあります。

- R→C(real-to-complex、実数から複素数へ)は2個の実数 x と y を1個の複素数(x、y)に変換します。
- C→R(complex-to-real、複素数から実数へ)は1個の複素数(x、y)を2個の実数 x と y に変換します。
- RE(real part、実数部分)は1個の複素数(x、y)からその実数部分の x を返します。

- IM(imaginary part、虚数部分)は 1 個の複素数(x、y)からその虚数部分のyを返します。
- CONJ(conjugate、共役)は 1 個の複素数(x、y)から共役の複素数(x、$-y$)を返します。
- SIGNは 1 個の複素数(x、y)からxとy軸方向の成分比($x / \sqrt{x^2+y^2}$、$y / \sqrt{x^2+y^2}$)を返します。

COMPLEXメニューの次の行に切り替えましょう。

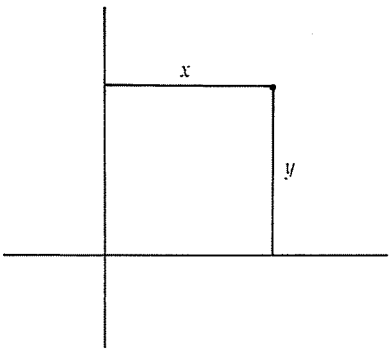
NEXT

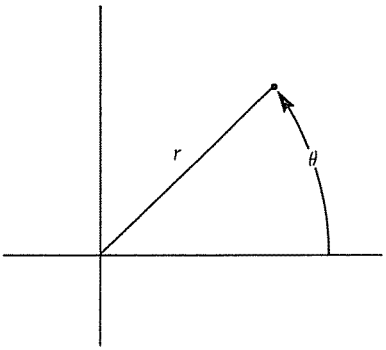


ここの関数(NEG 以外)は極座標系での複素数に関連したものです。

極座標系の使用法

平面上の 1 点は 2 種の別々な座標系で表すことができます。次図がその解説図で、直交座標系の表現(x、y)と極座標系の表現(r、θ)です。





- R→P(rectangular-to-polar、直交座標系から極座標系へ)は複素数を直交座標表現(x、y)から極座標表現(r、θ)に変換します。
- P→R(polar-to-rectangular、極座標系から直交座標系へ)は複素数を極座標表現(r、θ)から直交座標表現(x、y)に変換します。
- ABS(abusolute value、絶対値)は複素数の引き数(x、y)から動径rを返します。
- NEGは複素数の引き数(x、y)から正負符号を反転した($-x$、$-y$)を返します。
- ARG(argument、偏角)は複素数の引き数(x、y)から偏角θを返します。

P→Rだけが複素数を極座標系と解釈し、残りの全関数(算術、三角、対数、双曲線など)はどれも複素数を直交座標系と解釈します。次の規則が重要なので忘れないでください。すなわち極座標系の複素数は計算に使う前に直交座標系に変換することが必要です。

極座標系の算術計算例として、方位角36°で2マイル進み、次に方位角65°で3マイル進んだと仮定しましょう。結果の距離と方位角を小数点以下2桁で計算するとどうなるでしょうか？
角度を度単位に、数値表示をFIX2に切り替えましょう。

[MODE] [DEG] 2 [FIX]

最初の距離と方位角を入入力しましょう。

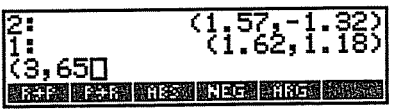
[(]2[,]36

直交座標系に変換しましょう。

[COMPLX] [NEXT] [P→R]

2番目の距離と方位角を入力しましょう。

[(]3[,]65



直交座標系に変換しましょう。

P→R

直交座標系で足し算をしましょう。

+

極座標系に変換しましょう。

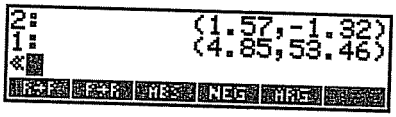
R→P

結果の距離は4.85マイル、方位角は53.46°です。

極座標の足し算用のユーザ定義関数

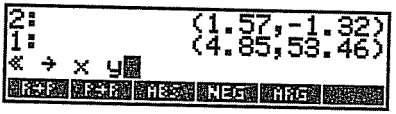
これは前の部分で手操作で計算したのを自動化する簡単なプログラムPSUM(polar sum、極座標の合計)です。
プログラムを開始しましょう。

«



引き数を示しましょう。(2個の引き数の間に空白を入れます。)

→ LC x SPACE y



右矢印はそれ以後の名称がローカル変数であることを示し、これはこのプログラム内部だけに通用する変数です。

関数を定義しましょう。

' R▸P (P▸R (x) ENTER
+ P▸R (y

閉じ括弧と識別記号が自動的に追加されました。

このプログラムの意味は次の通りです。スタックから(RPNを使ったとき)または数式内から(数式を使ったとき)から2個の引き数を取り出し、それをxとyと名付け、xとyの直交座標を合計して極座標化する。

このプログラムを変数PSUMに保存しましょう。

' PSUM STO

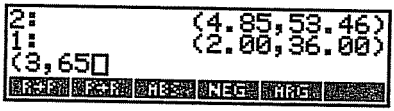
これからPSUMを使って、RPNを使った方法と数式を使った方法の両方で前の計算を繰り返してみましょう。

最初の距離と方位角を入力しましょう。

(2 , 36 ENTER

2番目の距離と方位角を入力しましょう。

(3 , 65

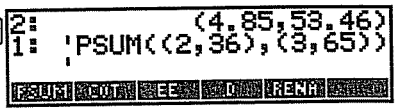


PSUMを実行しましょう。

USER PSUM

結果は前の答えと一致しています。
こんどは数式を使って試してみましょう。

' PSUM ((2 , 36) , (3 , 65) ENTER



外側の括弧と中心のコンマでPSUMの引き数を定義し、内側の括弧とコンマは複素数を表しています。数式中の引き数に複素数を使うときには2重の括弧が必要であることを忘れないでください。
数式を評価しましょう。

EVAL

第7章

プロット

この章ではHP-28Sのプロット機能を紹介します。プロット(グラフ化)すると、数式または方程式の性質を視覚的に理解するのに非常に役立ちます。それに、プロットを利用すると数式の根、極大、極小の推定が簡単になります。次の章でSOLVEを使って推定値から正確な数値にする方法を説明します。

この章ではPLOTメニュー内の一部のコマンドの使用法を説明します。PLOTメニューの全コマンドの要約は付録Dの「メニュー案内」のPLOTにあります。詳細はReference Manualの "PLOT" を見てください。

最初の例ではラジアン単位の角度で$\sin x$をプロットしますが、その前にHP-28Sの表示部を説明図に一致させるための準備があります。

プロットではプロット用のパラメータ(基準値)のリストを保存するためにPPARという名の変数を使います。既存のPPAR変数がない状態では自動的に初期値のPPARが作られます。次のプロットでは初期値のプロット用パラメータを使えるようにするための既存のPPARを削除しておきます。

スタックをクリアしてPLOTメニューに切り替えましょう。

CLEAR
PLOT

PLOTメニューの2行目に切り替えましょう。

NEXT

既存のPPARを削除しましょう。

' PPAR ▒ PURGE

角度単位をラジアンに、数値表示はSTD に切り替えましょう。

▒ MODE RAD STD

さて数式を入力しましょう。

' TRIG SIN X ENTER

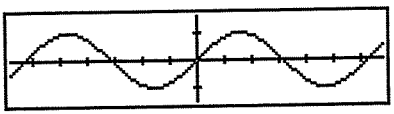
この数式をプロット専用変数(EQという特別な名称が付いている普通の変数)に保存しましょう。(これは第4章でSOLVR を使ったときと同じ規則です。)

▒ PLOT STEQ

STEQ を押すのは ' EQ STO と押すのと同じことです。

数式をプロットしましょう。

ORAW



案内表示の((●))が消えればプロットが完了した知らせです。

横線は独立変数(この場合は x)用の軸で、縦線は従属変数(数式sin x)の値用の軸です。両軸の目盛は長さ1を表します。

プロットの印字

HP 82240Aプリンタをお持ちの方は、次のようにするとプロットした画像をプリンタで印字することができます。

1. プリンタの説明書通りにプリンタを正しい位置に置きます。
2. [ON]を押したままにします。
3. [L](上側にPRINTと印刷してあるキー)を押します。
4. [ON]を放します。

このキー操作はキーボードのPRLCD(print LCD 、表示画面のプリント、これはPRINTメニューの1行目にあります)コマンドと同等です。このキー操作を使うと、計算操作に影響なくいつでも表示画面を印字できます。

数式をプロットしたあと自動的に結果を印字するプログラムを書くときには、コマンドの順序を次のようにしてください。

…CLLCD DRAW PRLCD…

この例に戻って、普通のスタックの表示に戻ることにします。

[ON]

プロット目盛の変更

一般的には、数式のプロットが最初からこのようにうまくできることはありません。数式のプロットに慣れるまでは、数式の特性をうまく表す画面を作るためにプロット範囲(これはプロット用パラメータで定義されます)の調整が必要になります。

プロット範囲を事前に知っていれば、PPAR内のプロット用パラメータを直接変更することもできます。(PPARの詳細はReference Manualの"PLOT"にあります。)目的のプロット範囲を見つけだすには、何回も実験が必要になることがあります。ここでは目的のプロット画面を得るために、PLOTメニューのコマンドをどう使うかを説明します。

2番目の例として、数式x^3-x^2-x+3をプロットしてみます。

スタック1段目に数式を入れましょう。

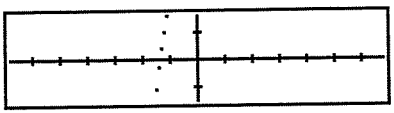
'X▼ ^ 3 − X▼ ^ 2 − X + 3
ENTER

この数式をプロット用数式として保存しましょう。

STEQ

数式をプロットしましょう。

DRAW



横線はx用の軸で、縦線は数式x^3-x^2-x+3の値用の軸です。

このプロットには数式のゼロ点(数式の値が0になるxの値)が見えます。このゼロ点は数式のグラフとx軸との交点です。次の章でSOLVRを使ってこのxを正確に求めます。

グラフを広く見るために、縦軸の範囲を広げてもう一度プロットします。

普通のスタック表示に戻しましょう。

ON

次のメニュー行にある ×H (times height、高さの倍率)操作を使って、縦軸の倍率を2倍にしてみましょう。

NEXT 2 ×H

新しいプロット用パラメータで、もう一度プロットしてみましょう。

PREV DRAW

横軸の目盛は間隔が1のままですが、縦軸の目盛は間隔が2に変わっています。

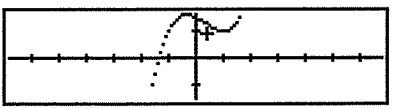
今度はプロットを移動して、重要部分を表示部の中心に動かすことにします。

プロットの中心位置変更

プロットすると表示部の中心に十字マークが現れます。(両軸の交点が表示部の中心にあるときには、この十字マークは見えません。)この十字マークを使って表示部内のどの点でも読み取って、スタックにその座標値を返すことができます。ここでは次のプロット時の中心点にしたい点を読み取り、これをプロット用パラメータの調整に使いましょう。

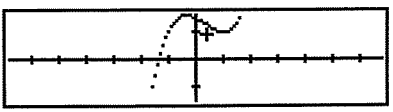
十字マークを図中の位置に動かしましょう。

▶ (4回押します)
▲ (3回押します)



点を読み取りましょう。

INS



スタック表示に戻りましょう。

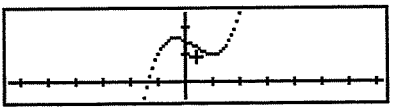
ON

読み取った点の座標値は、複素数の形になって、1段目に入っています。
次のメニュー行の CENTR を使って、プロットの中心点を定義しましょう。

NEXT CENTR

この座標値はスタックから取り出されて、プロット用パラメータの調整に使用されました。 ZOUT とは違って、 CENTR は目盛を変えません。
もう一度プロットしてみましょう。

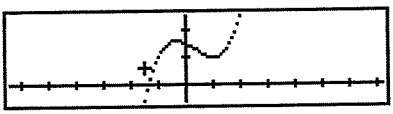
PREV ERASE



今度はプロットの重要部分を拡大しましょう。小数値の目盛倍率を使って、もう一度 ZOUT を使うこともできます。(例えば、.5の目盛倍率を使うと縦軸の目盛が当初の値に戻ります。)しかしプロットを拡大するにはもっと応用範囲の広い柔軟な方法があります。

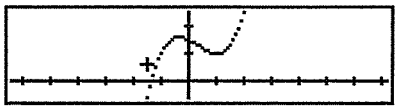
プロット範囲の指定

プロットを拡大するために、今度は2点を読み取ります。1点は次のプロットで左下隅にしたい点で、もう1点は右上隅にしたい点です。
十字マークを目的の左下隅に動かしましょう。

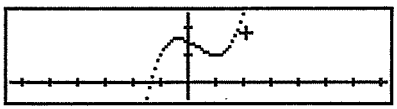


この点を読み取りましょう。

INS

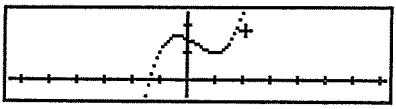


十字マークを目的の右上隅に動かしましょう。



この点を読み取ります。

INS



スタック表示に戻りましょう。

ON

左下隅の座標値は、複素数の形になっていて、2段目に入っています。右上隅の座標値は1段目に入っています。(実際の座標値は図中のと少し違っていることもあります。)

PMAX (plot maxima、プロットの最大点)を使って、プロットの右上隅を定義しましょう。

PMAX

この座標値はスタックから取り出されて、プロット用パラメータの調整に使用されました。

PMIN (plot minima、プロットの最小点)を使って、プロットの左下隅を定義しましょう。

PMIN

もう一度プロットしてみましょう。

DRAW

プロットの高さと幅を変えたので、縦軸と横軸の両方の目盛の単位が変わっていることに注意してください。

このプロットで数式のグラフに2個の変極点があることがわかります。1個は極大値で、残り1個が極小値です。次の章でSOLVERを使って極小値の正確な値を求めます。このプロット用パラメータを作る操作を繰り返さなくてもすむように、PPARの値を別な名称の変数に保存しておきましょう。次の章でプロットを再現するためには、この値をPPARに戻すだけでよいことになります。

スタック表示に戻りましょう。

ON

PPARの内容をスタックに入れましょう。

NEXT PPAR

プロット用パラロータの説明とプロット全般についての詳細は、Reference Manualの"PLOT"を見てください。

このプロット用パラメータが入っている変数PPAR1を作りましょう。

' PPAR 1 STO

これでSOLVERを使って数式のゼロ点と極小値になる正確な値を求める準備ができました。

等式のプロット

この章での例はどちらも数式でしたが、両辺を＝で結んだ等式のプロットにも同じ規則と方法が使えます。変数EQ内が等式のときには、DRAWでは等式の両辺を2つの数式としてプロットします。2個のグラフの交点を見つけることで等式の根を見つけることができます。その点が等式の両辺の値が同じになる点だからです。

第8章

SOLVER(多変数式解答機能)

この章では、前の章でプロットした数式のゼロ点と極小値を与える点を求める方法を説明します。前の章の結果の一部を使うので、まだやったことがない方は、前の章の通りに操作しておいてください。

SOLVERの詳細は、Reference Manualの"SOLVE"を見てください。

数式のゼロ点を求める方法

これからの例では、前の章で説明した通りに数式 x^3-x^2-x+3 がSOLVER用の演算式になっていて、変数PPAR1を作ってあるものと仮定しています。数式を再びプロットして、数式の値が0になる点を読み取り、次にSOLVERを使ってこの値をもっと正確に求めます。

次の例を始める前に、スタックをクリアし、角度をラジアン単位に、表示形式をFIX 2に切り替えましょう。

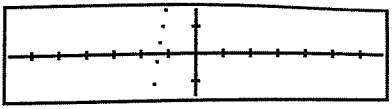
[CLEAR] [MODE] RAD
2 FIX

次のプロットでは念のため、プロット用パラメータの初期値を使えるように、既存のPPARを削除しましょう。

[PLOT] [NEXT]
' PPAR [PURGE]

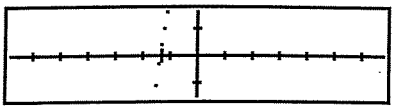
数式をプロットしましょう

[PREV] [DRAW]



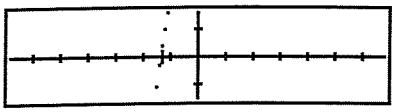
このプロットで数式のゼロ点値(数式の値が0になるxの値)がわかります。この0は数式のグラフと横軸との交点です。

十字マークをグラフと横軸の交点の近くに動かしましょう。(十字マークを動かすには[▲]、[▼]、[◀]、[▶]を使います。)



この点を読み取りましょう。

[INS]

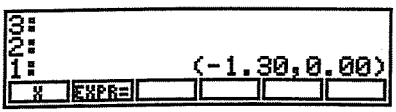


この点を推定値として使って数式のゼロ点を正確に求めることにします。(数式に2個以上の根がある場合には、この推定値で求めたい点を指定します。)
スタック表示に戻りましょう。

[ON]

読み取った座標点は、複素数の形で、1段目に入っています。(読み取った座標点が図とは少し違うこともあります。)
SOLVRメニューに切り替えましょう。

[SOLV] [SOLVR]

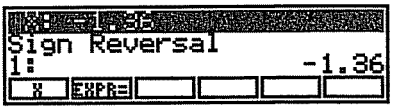


この例では変数は x だけですが、一般的には、このSOLVER専用変数内の全変数がSOLVRメニューに現れます。
読み取った推定値を変数 x に保存しましょう。

[X]

読み取った座標点には2個の数値が入っていますが、SOLVERは推定値として最初の数値だけを使います。
それでは x を解きましょう。

[X]



メッセージの `Sign Reversal` は、SOLVERが12桁まで正しい近似値を見つけたことを表しています。SOLVERがぴったり正確な解をみつけると、表示するメッセージは `Zero` になります。このメッセージを、解釈用メッセージと呼び、Reference Manualの〝SOLVE〟に説明があります。
普通のスタック表示に戻りましょう。

[ON]

極大値または極小値を求める方法

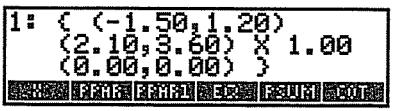
数式のゼロ点を求めるときは、初めの推定値を出発点としてSOLVERが次第に x 軸により近い点を探索しました。正の極小値または負の極大値にごく近い点を推定値にすると、x 軸にそれより近い点を見つけることはできません。このような場合には、SOLVERは0でなくて極値(極小または極大値)を見つけます。(一般的に言うと、SOLVERに極値を求めるように仕向けたとき以外は、SOLVERが極値を捕えることはありません。)

96ページの前章で作ったグラフを見てください。このグラフからこの数式には正の極小値と正の極大値が1個ずつあることがわかります。極小値はある範囲内でx軸に一番近い点なので、SOLVERで極小値を求めることができます。しかし極大値はある範囲内でx軸に一番遠い点なので、SOLVERでは極大値を求めることはできません。

ここでは変数PPAR1内に保存したプロット用パラメータを使って数式をプロットし、極小値を推定するための3点を読み取り、次にSOLVERを使ってもっと正確な極小値を求めることにします。

PPAR1内に保存したリストをスタックに戻しましょう。

USER PPAR

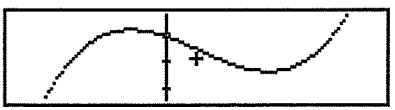


PPAR1に内に保存してある値を変数PPARに再保存しましょう。

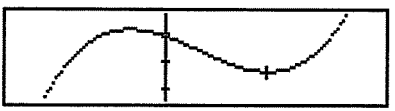
' PPAR STO

数式をプロットしましょう。

PLOT ERASE

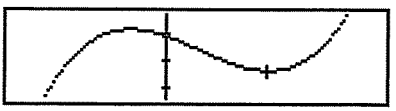


十字マークをほぼ極小値に近いところに動かしてください。

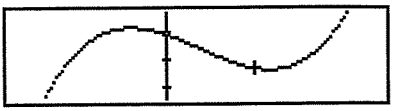


その点を読み取りましょう。

INS

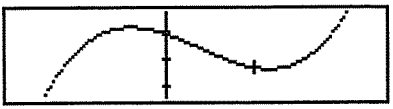


十字マークを極小値の左隣に動かしてください。



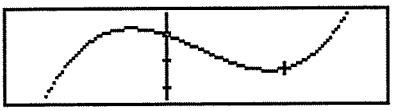
この点を読み取りましょう。

INS



十字マークを極小値の右隣に動かして、その点を読み取りましょう。

INS



スタック表示に戻りましょう。

ON

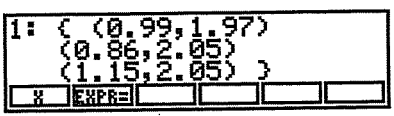
3点が1、2、3段目に入っています。(図中の数値とは少し違うこともあります。)

次にこの3個の概算値を1個のリストに組み合わせましょう。こうすることで、3個の概算値を1個のオブジェクトとして扱うことができます。これがリスト(いくつかのオブジェクトを1個にまとめたもの)の典型的な使用法です。

■ LIST 3 →LIST

SOLVERメニューに切り替えましょう。

SOLV SOLVR



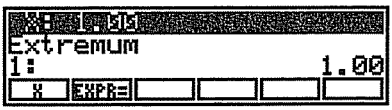
この例では変数はxだけですが、一般的にはこのSOLVER用演算式内の全変数がSOLVERメニューに現れます。

座標点のリストを変数xに保存しましょう。

[X]

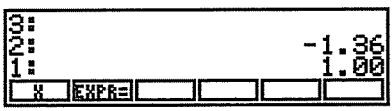
スタックから座標点のリストが取り出され、初期推定値として変数x内に保存されました。
xを解きましょう。

[SOLVE] [X]



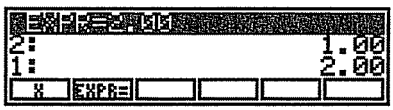
メッセージのExtremumは、SOLVERが数式の極値点を見付けたことを表しています。
普通のスタック表示に戻りましょう。

[ON]



極値を算出しましょう。

[EXPR=]



極値は2です。

均等払い複利計算

ここでは均等払い複利計算(TVM、time value of mony)にSOLVERをどう使うかを説明します。期間数をn、単位期間の利率をi%、支払額をpmt(円またはドル、以下同様)、現価をpv、終価をfvとすると、TVMの等式は次のようになります。(現価とは計算開始時の金額で、預貯金では元金、ローンではローン対象額に当たります。終価とは計算終了時の金額で、預貯金では元利合計、ローンでは最終回一括返済額に当たります。)

$$(1-\text{sppv})\times\text{pmt}\times(100/i)+\text{pv}=-f\text{v}\times\text{sppv}$$

この式中のsppvは複利現価係数で、次の式になります。

$$\text{sppv(single payment present value)} = (1 + i/100)^{-n}$$
$$= \exp(-n \times \ln(1 + i/100))$$

この公式は各期の期末に支払いが発生するものと仮定しています(期末払いと呼びます)。

主要手順は次の通りです。

1. sppvの数式をキー入力して、これを変数SPPVに保存します。
2. 公式をキー入力して、これを変数TVMに保存します。
3. TVMを変数EQに入れます。
4. SOLVRを使って5個の変数n、i、pmt、pv、fvのどれか4個に代入してから、残り1個を求めます。

開始前に、スタックをクリアし、数値表示形式をFIX 2にしておきましょう(アメリカでは通貨の補助単位としてドルの1/100のセントがあるからです)。

[ON] [CLEAR]
[ON] [MODE] 2 [FIX]

手順1. sppvの数式をキー入力して、これを変数sppvに保存します。

sppvの数式をキー入力しましょう。

['] [ON] [LOGS] [EXP] [−] N [×]
[LNP1] [(] 1 [÷] 100 [)] [ENTER]

この数式はLNP1を使っているので、ln(1 + i/100)を計算するよりもよい精度になります。

変数SPPVを作って、USERメニューを調べてみましょう。

['] [ON] SPPV [STO]
[USER]

手順2. 等式をキー入力して、これを変数TVMに保存します。
TVMの等式をキー入力しましょう。

['] [( ] 1 [−] [SPPV] [ )] [×] PMT
[×] 100 [÷] I [+] PV [=] [−] FV [×]
[SPPV] [ENTER]

変数TVMを作りましょう。

['] TVM [STO]

USERメニューに新しくTVMのメニュー記号が追加されました。

手順3. TVMを変数EQに入れます。
名称のTVMをキー入力しましょう。

['] [TVM]

名称TVMを変数EQに保存しましょう。

[SOLV] [STEQ]

手順4. SOLVERを使って5個の変数n、i、pmt、pv、fvのどれか4個に代入してから、残り1個を求めます。
SOLVRメニューに切り替えましょう。

[SOLVR]

TVMとSPPV内の全部の変数がメニュー内に見えます。(SPPV内の変数も見えるのは、SOLVER用変数内のTVM内にSPPVが入っているからです。)

年利が11.5%、30年間のローンで、毎月の支払いが630ドル、最終回一括支払額が0とすると、ローンは最高いくらまで借りることができるでしょうか？
N＝30×12、I＝11.5／12、PMT＝−630、FV＝0を代入してPVを求めましょう。(PMTが負数なのは、上の等式で支払いは負数、受領は正数と決めてあるからです。)

まずNに値を代入しましょう。

30 [ENTER] 12 [×] [N]

Iに値を代入しましょう。

11.5 [ENTER] 12 [÷] [I]

PMTに値を代入しましょう。

−630 [PMT]

FVにも値を代入しましょう。

0 [FV]

今度はPVを求めましょう。

[■] [PV]

63617.64ドルまで借りることができます。メッセージの Zero は、SOLVE専用変数を満足するような正確な値が12桁以内で返せたという意味です。

第9章

代数を含む解の求め方

この章では代数が入ったままの答えを求める2種の方法を説明します。最初の方法は2つのゼロ点を表す1次式を利用して2次式を解く方法です。2番目は一般的な等式内の1個の変数について、代数が入ったままの答えを求めるという、もっと応用範囲の広い方法です。

どちらの方法も数式と等式の両方に使えます。数式$f(x)$のゼロ点とは等式$f(x)=0$の根と同じで、等式$f(x)=g(x)$の根とは数式$f(x)-g(x)$のゼロ点と同じことです。

二次式のゼロ点を求める方法

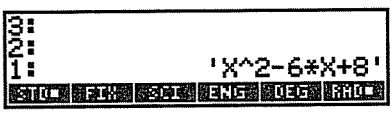
プロットや推定をしなくても二次式の両方のゼロ点を求めることができます。次の例ではx^2-6x+8を解きます。

例題を始める前に、スタックをクリアし、数値表示形式をSTDに切り替えます。

[ON] [CLEAR]
[ON] [MODE] [STD]

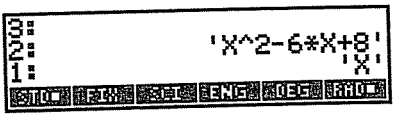
数式をスタックに入れましょう。

[']X[α] [^]2 [−]6 [×]X [+]8 [ENTER]



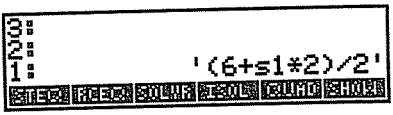
変数名xをスタックに入れて、求めたい変数を指定しましょう。

' X ENTER



SOLVEメニュー内のQUAD(quadratic、二次式)を使って、ゼロ点を計算しましょう。

SOLV QUAD

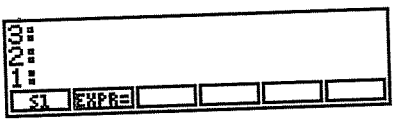


この数式で二次式の両方の根を表しています。変数s1は任意の符号(＋１または－１のどちらか)を表し、数式の根はs1にどちらかの値を入れたときの値です。
この数式を変数EQに保存します。

STEQ

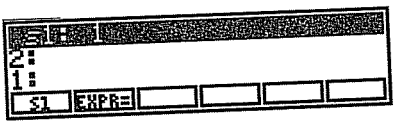
SOLVERメニューに切り替えましょう。

SOLVR



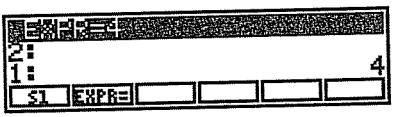
この数式ではs1だけが変数です。
まずs1を正の符号にしましょう。

1 S1



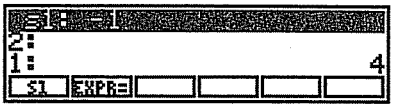
１番目の答えをスタックに返しましょう。

EXPR=



今度はs1を負の符号にしましょう。

−1 [S1]



2番目の答えをスタックに返しましょう。

[EXPR=]

x^2-6x+8の2個の根は$x=4$と$x=2$です。

変数の分離

HP-28Sは等式内に1回だけ現れる1個の変数を分離して、等式の記号解を表す数式を返すことができます。例えば、等式で求めたい変数がxで、その等式の他の変数がa、b、cだとすると、xの分離とはその等式を変形させて、a、b、cでxを表す数式を作り出すことを言います。

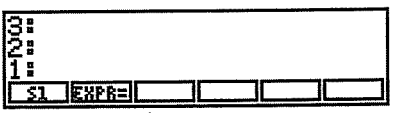
最初の例として、次の等式のxを分離します。

$$a(x+3)-b=c$$

この例はxが1回だけなので簡単です。後の例では変数が1回だけになるように等式を処理する方法を説明します。

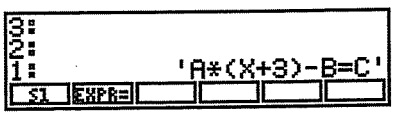
スタックをクリアしましょう。

[CLEAR]



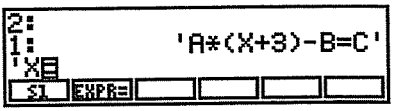
等式をスタックに入れましょう。

['] A [×] [(] X [+] 3 [)] [−] B [=] C
[ENTER]



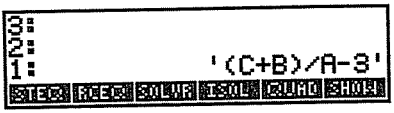
分離したい変数を指定しましょう。

[']X



SOLVEメニュー内のISOL(isolate、分離)を使って、xを分離しましょう。

[SOLV] [ISOL]



返された数式は等式のxについての記号解です。つまり$x=(c+b)/a-3$のときに等式

$$a(x+3)-b=c$$

が成り立ちます。

式の展開とまとめ

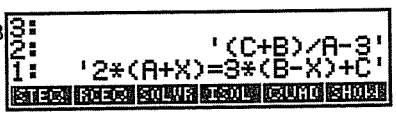
xが2回以上あるときには、xが1回だけになるように等式を処理することが必要となります。2番目の例で次の等式内のxをどのように分離するかを見てください。

$$2(a+x)=3(b-x)+c$$

この例を解く方法は、まず等式を展開し、どちらか一変のx項を両辺から引いて、一辺のx項を取り消し、他辺のx項を1個にまとめ、最後にそれを分離します。

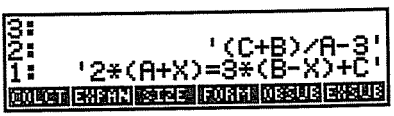
等式をスタックに入れましょう。

['] 2 [×] [(] A [+] X [)] [=] 3 [×] [(] B [−] X [)] [+] C [ENTER]



ALGEBRAメニューに切り替えましょう。

[ALGEBRA]



この例では等式を処理するのに EXPAN (expand、展開)と COLCT (collect、同類項のまとめ)を使います。3番目の例では等式の処理に FORM (form algebraic expression、数式エディタ)を使います。ALGEBRAメニュー内の全コマンドの要約は付録Dの「メニュー案内」にあります。詳しい説明はReference Manualの〝ALGEBRA〟を見てください。それに、FORMは強力な数式エディタなので、Reference Manualにはこれだけを取り上げた〝ALGEBRA(FORM)〟があります。

等式の両辺を展開しましょう。

EXPAN

```
3:
2:                    '(C+B)/A-3'
1:    '2*A+2*X=3*B-3*X+C'
COLCT EXPAN SIZE FORM DESUB EXSUB
```

等式の両辺から左辺の x 項(2 x)を引くために、まず左辺の x 項をスタックに入れましょう。

' 2 × X ENTER

```
3:                    '(C+B)/A-3'
2:    '2*A+2*X=3*B-3*X+C'
1:                          '2*X'
COLCT EXPAN SIZE FORM DESUB EXSUB
```

次に両辺から2 x を引きましょう。

−

```
2:                    '(C+B)/A-3'
1:  '2*A+2*X-2*X=3*B-3*X
   +C-2*X'
COLCT EXPAN SIZE FORM DESUB EXSUB
```

等式をまとめましょう。

COLCT

```
3:
2:                    '(C+B)/A-3'
1:              '2*A=3*B+C-5*X'
COLCT EXPAN SIZE FORM DESUB EXSUB
```

両辺を別々にまとめたので、左辺の x 項が消えました。

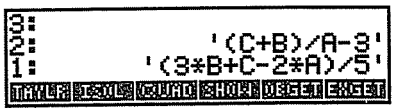
これど等式内の x が分離できます。分離したい変数を指定しましょう。

' X

```
2:                    '(C+B)/A-3'
1:              '2*A=3*B+C-5*X'
'X
COLCT EXPAN SIZE FORM DESUB EXSUB
```

xを分離しましょう。SOLVEメニュー内と同じISOLコマンドがALGEBRAメニューの2行目にもあります。

NEXT　ISOL



この返された式が等式のxについての記号解です。つまり、次の等式

$$2\,(a+x)=3\,(b-x)+c$$

は$x=(3b+c-2a)/5$のときに成り立ちます。

FORM(数式エディタ)の使用法

xが何回もあって、そのどれかに代数の係数がついていると、COLCTコマンドでは係数を組み合わせることができません。3番目の例として次の等式のxを分離しますが、xは2回あって最初のものに代数の係数aが付いています。

$$a(x+b)+2x=c$$

手順は等式を展開し、FORMを使ってxの係数をまとめ、最後にxを分離するということになります。

等式をスタックに入れましょう。

' A × (X + B) + 2 × X = C ENTER

等式を展開しましょう。

NEXT　EXPAN

今度はFORMを使ってxの係数をまとめましょう。

FORM

FORMが働いている間は普通の計算操作ができません。FORMで表示する等式は個々の部分式をどれも括弧で囲んでいます。(部分式とは数式内の部分的な数式のことです。)これからFORMを使って等式内の部分式を処理しましょう。

目標は〈A*X〉と〈2*X〉を単項の〈〈A+2〉*X〉にまとめることです。筆算でやるときのように、次の3手順が必要です。等式の現在の形は次のようになっています。

$$(ax+ab)+2x=c$$

最初にaxとabの位置を交換して、次のようにします。

$$(ab+ax)+2x=c$$

2番目にaxと$2x$を結合して、次のようにします。

$$ab+(ax+2x)=c$$

3番目にaxと$2x$を併合して、次のようにします。

$$ab+(a+2)x=c$$

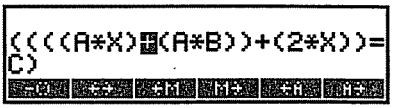
手順1. axとabを交換します。

カーソル(白抜き文字、つまり黒白が逆になっている文字)を＋に動かしましょう。

カーソルの位置は実際に操作する部分式によって決まります。ここが部分式〈〈A*X〉+〈A*B〉〉の＋の引き数を入れ替えたい位置です。

＋の処理をするため次の行に切り替えましょう。

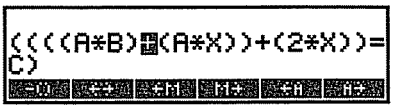
NEXT



NEXT を押したときに現れる処理用のメニューは、カーソル位置の関数または変数によって変わります。これは＋専用のメニューです。

COMMUTE (commute、交換)を使って、＋の引き数を交換しましょう。

COMMUTE



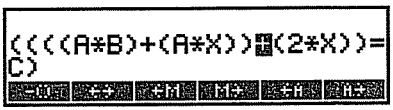
FORMの主メニューに戻りましょう。

ENTER

手順2． ax と2x を結合します。
カーソルを2番目の＋に動かしましょう。

ここが1個の部分式 ((((A*B)+(A*X))+(2*X)) の中で項 (A*X) と (2*X) を結合させたい位置です。＋の処理用の1行目に切り替えましょう。

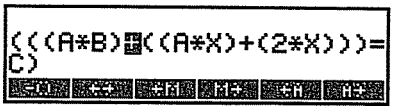
NEXT



カーソルがまた部分式の足し算位置にあるので前回と同じ処理内容です。

A→ (associate right、右側に結合)を使って、項 ‹A*X› と ‹2*X› を結合して部分式 ‹‹A*X›+‹2*X›› にしましょう。

A→



FORMの主メニューに戻りましょう。

ENTER

手順3． ax と2x を併合します。
カーソルを2番目の＋に動かしましょう。

←→ ←→ ←→ ←→

ここが部分式 ‹‹A*X›+‹2*X›› で X の係数を組み合わせたい位置です。
＋の処理用の1行目に切り替えましょう。

NEXT

→MRG (merge right、右側に併合)を使って、X の係数を組み合わせましょう。

→MRG

これで ‹A*X› と ‹2*X› を組み合わせて、単項の ‹‹A+2›*X› にするという目標に到着しました。
FORMから抜け出して、変更した等式をスタックに返しましょう。

ON

これで等式内に現れるxが1回だけになったので、xを分離することができます。

分離したい変数を指定しましょう。

[']X

xを分離しましょう。

NEST

この返された式が等式のxについての記号解です。つまり、次の等式

$$a(x+b)+2x=c$$

は$x=(c-ab)/(a+2)$のときに成立します。

第10章

微積分

対応する導関数が存在するものならどんな数式でも代数式のままで微分できます。積分にはかなりの制約があって、数値定積分はどんな数式でも計算できますが、代数式のままで積分を求めることができるのは多項式だけです。

この章には、導関数、不定積分、定積分を求める簡単な例があります。HP-28Sでの微積分の詳細は、Reference Manualの〝Calculus〟にあります。

数式の微分

微分の規則を計算機がどう適用するかを観察しながら、1段階ずつ数式を微分することも、一度にまとめて数式を微分することもできます。最終結果はどららでも同じです。ここでは同じ数式の微分を2回します。初めは段階を追って、次は一度にまとめて微分します。

1段階ずつの微分

段階を追って微分するために、数式を導関数の形でキー入力しましょう。この例では、次の式を計算します。

$$\frac{d}{dx}\tan(x^2+1)$$

この例を始める前に、スタックをクリアし、角度単位をラジアンに、数値表示形式をSTD に切り替えましょう。

CLEAR
MODE
RAD
STD

変数x内に数値があると答えも数値になってしまうので、これを削除しておきましょう。

' X PURGE

これから導関数の数式のキー入力を始めますが、最初は微分する変数です。

' d/dx X (

次は微分する数式のキー入力です。

TRIG TAN X ^ 2 + 1 ENTER

これがtan(x^2+1)のxについての導関数を表した数式です。
この数式を1回評価しましょう。

EVAL

この結果は合成関数の微分法を反映したものです。

$$\frac{d}{dx}\tan(x^2+1)=\frac{d}{d(x^2+1)}\tan(x^2+1)\times\frac{d}{dx}(x^2+1)$$

tanの導関数の評価はすんでいます。次はx^2+1の導関数の評価です。

数式の2回目の評価をしましょう。

EVAL

この結果は次の和の導関数を反映したものです。

$$\frac{d}{dx}(x^2+1)=\frac{d}{dx}(x^2)+\frac{d}{dx}1$$

1の導関数は0なので、この項は消えてしまいました。次はx^2の導関数です。

数式の3回目の評価をしましょう。

EVAL

この結果は次の定理を反映したものです。

$$\frac{d}{dx}x^2=\frac{d}{dx}(x)^2\times\frac{d}{dx}x$$

x^2の導関数の評価はすんでいます。最後は、xそのものの導関数の評価です。

数式の4回目の評価をしましょう。

EVAL

これで導関数の評価が終わりました。

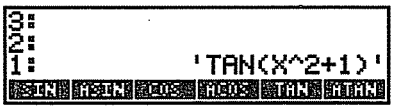
一度で微分する方法

数式を一度の操作で微分するには、スタック操作の方法で微分を実行します。微分する式は次の通りでした。

$$\frac{d}{dx}\tan(x^2+1)$$

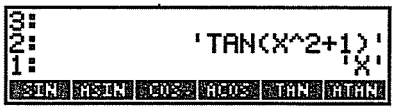
微分する数式をスタックに入れましょう。

▒ CLEAR
' TAN X ▒ ^ 2 + 1 ENTER



微分する変数を指定しましょう。

' X ENTER



数式を微分しましょう。

▒ d/dx

完全に評価が終わった導関数が1段目に返りました。

数式の積分

HP-28Sは数式の不定積分を代数式のままの積分として計算し、結果を数式で返します。この方法で正確な結果が返るのは多項式だけです。(これ以外の数式では、HP-28Sはテイラー級数の近似として積分します。詳細はReference Manualの"calculus"を見てください。)次のページの最初の例で代数式のままの積分を見ます。

対照的に、定積分は数値積分として計算し、数値の結果を返します。この方法は数学的に意味があるならどんな数式にも使えます。2番目の例で数値積分を見ます。

代数式のままで多項式の積分

この例では次の多項式を代数式のままで積分します。

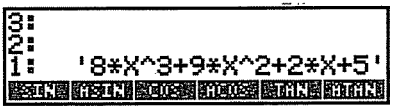
$$8x^3+9x^2+2x+5$$

スタックをクリアしましょう。

▒ CLEAR

多項式をスタックに入れましょう。

' 8 × X ▒ ^ 3 + 9 × X ▒ ^ 2
+ 2 × X + 5 ENTER

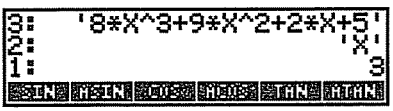


積分する変数を指定しましょう。

' X ENTER

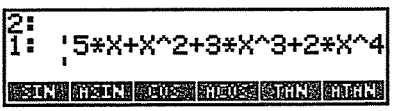
多式の次数を指定しましょう。

3 ENTER



多項式を積分しましょう。

▒ ∫



案内表示の((●))が消えるまで待ってください。消えると積分が完了です。積分結果はスタックは1段目に返ります。

数式の数値積分

この例は次の数式の数値積分値を求めます。

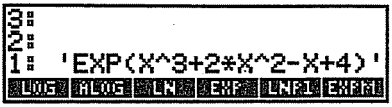
$$\exp(x^3+x^2-x+4)\,dx$$

スタックをクリアしましょう。

▒ CLEAR

数式をスタックに入れましょう。

' ▒ LOGS ▒▒▒▒ X ▒ ^ 3 +
2 × X ▒ ^ 2 − X + 4 ENTER



変数名と積分区間をキー入力しましょう。この3個のオブジェクトを1個のリスト内に入れて入力しましょう。(これがリストの典型的な利用法で、何個かのオブジェクトを組み合わせたものを1個のオブジェクトのように扱うことができます。)

{ X SPACE 0 SPACE 1 ENTER

xは積分する変数、0と1は積分区間です。

次に希望する精度をキー入力します。

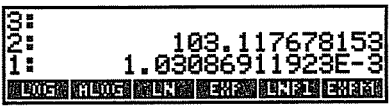
数式内に実験データから誘導した定数が入っている場合は、その定数の精度を指定してください。例えば、定数が小数点以下3桁まで正確なら、精度として.001を指定します。

この例で積分する数式には実験的定数がないので、12桁の精度まで指定することができます。しかし、精度を上げるほど数値積分の繰り返し計算に必要な時間が長くなるので、ここでは.00001の精度を指定しましょう。

1E−5 ENTER

積分値を求めましょう。

▒ ∫



積分の概算値が2段目に返り、誤差の上限が1段目に返りました。

積分値は103.118 ±.001です。返ってきた誤差の上限は積分の概算値と指定した精度の積に近似しています。

第11章

ベクトルと行列

HP-28Sは2種の配列が扱えます。ベクトルは1次元の配列で、行列は2次元の配列です。ベクトルや行列を1個のオブジェクトとして入力することができ、これを配列オブジェクトと呼び、数値と同じように簡単に計算処理ができます。

この章では実数配列(ベクトルと行列でその要素が実数のもの)を使った配列の基本計算を取り上げます。同様に要素が複素数の配列でも計算できます。

ARRAY メニューの全コマンドの要点は付録Dの「メニュー案内」にあります。詳細はReference Manualの〝ARRAY〟を見てください。

ベクトル

ここではベクトルの四則計算と外積、内積を取り上げます。

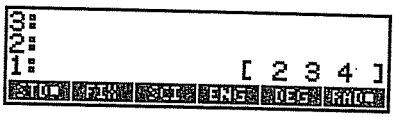
ベクトルのキー入力

これからの3例を始める前に、スタックをクリアし、数値表示形式をSTD に切り替えましょう。

[CLEAR]
[MODE] STD

ベクトル［2　3　4］をキー入力しましょう。2と3の間と3と4の間をコンマ(基数記号でない方)、または空白(SPACE)で分離します。

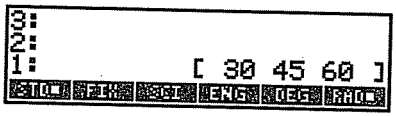
[ ] 2,3,4 ENTER



1個のベクトルと1個の数値の掛け算と割り算

このベクトルに15を掛けましょう。

15 ×



2個の数値の掛け算で引き数の順序による結果の違いがなかったように、この掛け算でも引き数の順序による違いはありません。しかし、割り算では、ベクトルは必ず2段目に、数値は1段目に入れておきます。
このベクトルを5で割りましょう。

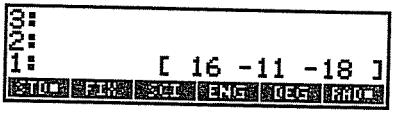
5 ÷

ベクトルどうしの足し算と引き算

2個の数値の足し算や引き算のように、要素数が同しベクトルであれば、2個のベクトルの足し算と引き算ができます。引き算では、ある数値から別の数値を引くときにその順序が大切であったように、引き数の順序が大切です。

この例では、上のベクトルからベクトル［−10　20　30］を引きましょう。

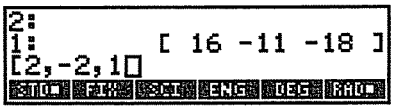
[ ] −10,20,30 −



外積の計算

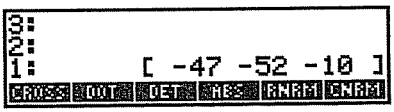
1段目のベクトルとベクトル［2　－2　1］の外積(cross)を求めましょう。(外積は要素数が2個または3個のベクトルにだけ定義されています。)
ベクトルをキー入力しましょう。

[ ] 2,−2,1



ARRAYメニューの3行目にあるCROSSを使って、外積を計算しましょう。

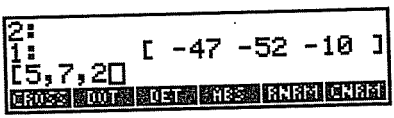
▒ ARRAY　NEXT　NEXT　CROSS



内積の計算

1段目のベクトルとベクトル［5　7　2］との内積(Dot)を求めましょう。
(2個のベクトルの要素数が等しくなければなりません。)
ベクトルをキー入力しましょう。

[ ] 5,7,2



内積を計算しましょう。

DOT

行列

ここでは逆行列と行列式を求める方法を説明します。どちらの計算も正方行列(行数と列数が同じ行列)に限定されます。

ベクトルに対して実行した計算は行列にも適用できます。(ただし、内積と外積は除きます)。１個の行列と１個の数値との掛け算と割り算もできますし、２個の行列の大きさが同じ(互いの行数と列数が同じ)であれば足し算と引き算もできます。

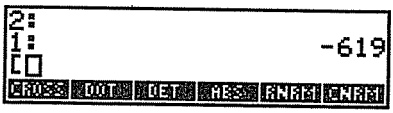
行列のキー入力

次の行列をキー入力しましょう。

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 3 \\ 1 & 2 & 4 \end{bmatrix}$$

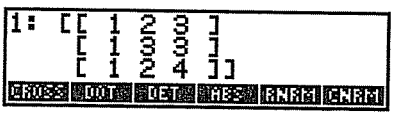
行列を開始しましょう。

[[ ]



行列の各行ごとに、別々のベクトルのように入力します。

[[ ] 1,2,3
[[ ] 1,3,3
[[ ] 1,2,4 ENTER



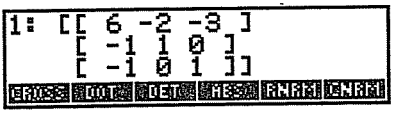
大きな行列を見る方法

要素数が多い、または要素が整数でない行列では、行列全体を一度で見ることができません。大きい行列を見るには ▒ EDIT (行列が１段目にあるとき)またはn ▒ VISIT (行列がn段目にあるとき)を使って、その行列をコマンド行に戻します。それからカーソル機能キーを使うと、行列のどの部分でも見ることができます。詳細は、第18章の「既存のオブジェクトの修正」を見てください。

逆行列

1段目の行列が正方行列なので、その逆行列を求めることができます。

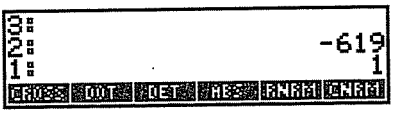
[shift] [1／x]



行列式

1段目の行列が正方行列なので、その行列式(Determinant)を求めることができます。

[DET]



2個の配列の掛け算

2個の行列どうし、または1個の行列と1個のベクトルとの掛け算には [×] が使えます。(2個のベクトルどうしの掛け算には、126ページのように [CROSS] または [DOT] を使います。)

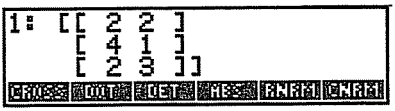
2個の行列の掛け算

2個の行列の掛け算では引き数の順序が重要です。2段目の行列の列数と1段目の行列の行数が同じであることが必要です。例えば、次の行列の積は計算できます。

$$\begin{bmatrix} 2 & 2 \\ 4 & 1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 2 & 2 & 1 & 4 \\ 3 & 4 & 2 & 1 \end{bmatrix}$$

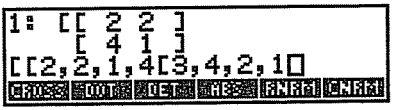
この行列の積を計算するには次のようにします。
最初の行列を入力します。

[ ] [ ] 2,2
[ ] 4,1
[ ] 2,3 ENTER



2番目の行列をキー入力します。

[ ] [ ] 2,2,1,4
[ ] 3,4,2,1



行列を掛けましょう。

×

1個の行列と1個のベクトルとの掛け算

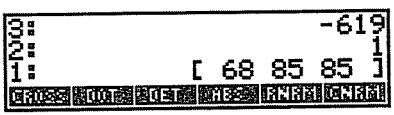
1個の行列と1個のベクトルとの掛け算のときにも、引き数の順序が重要です。行列は2段目に入っていて、ベクトルは1段目に入っている必要があります。そして行列の列数がベクトルの要素数と同じであることが必要です。

次の例では、1段目に入っている行列にベクトル［3　1　1　2］を掛けます。
ベクトルをキー入力しましょう。

[ ] 3,1,1,2

行列とベクトルを掛けましょう。

×



連立一次方程式の解

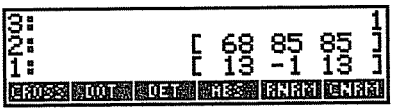
変数がn個のn元連立一次方程式を解くには、要素がn個の定数ベクトル、n×nの係数行列、それに [÷] を使います。定数ベクトルは方程式の定数値が入っているものです。係数行列は変数の係数が入っているものです。

次の例では3個の変数が入っている3元連立一次方程式を解きます。方程式は次の通りと仮定します。

$$\begin{aligned} 3x + y + 2z &= 13 \\ x + y - 8z &= -1 \\ -x + 2y + 5z &= 13 \end{aligned}$$

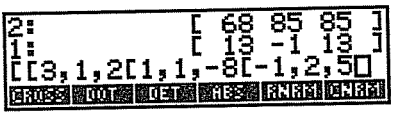
定数ベクトルを入力しましょう。

[ ] 13,−1,13 ENTER



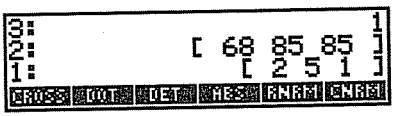
係数行列をキー入力しましょう。

[ ] [ ] 3,1,2
[ ] 1,1,−8
[ ] −1,2,5



連立方程式を解きましょう。

[÷]



解ベクトル内の値が連立方程式を成立させる変数の値です。

$$x = 2,\ y = 5,\ z = 1$$

方程式の数が少ない、または方程式の数が多い、あるいは特異行列に近い連立一次方程式を解くにはReference Manualの〝ARRAY〟を見てください。

第12章

統計計算

この章では、STATメニュー内のコマンドを使って、統計データの入力方法と、1変量統計と2変量統計の計算法を説明します。STATメニュー内の全コマンドの要約は付録Dの「メニュー案内」にあります。詳細は、Reference Manualの〝STAT〟を見てください。

次の表はアメリカの5年間の消費者物価指数(CPI)、生産者物価指数(PPI)、失業率(UR)をパーセントで表したものです。このデータを入力して、この基本統計量を計算してみましょう。

統計例題用のデータ

年	CPI	PPI	UR
1975	9.1	9.2	8.5
1976	5.8	4.6	7.7
1977	6.5	6.1	7.0
1978	7.6	7.8	6.0
1979	11.5	19.3	5.8

データの入力

統計データはΣDATという名の統計用行列に保存します。(普通の行列ですが、統計計算で使うことに決めてある行列です。)行列の各行に項目ごとのデータが1個ずつ入ります。この例では1行に1年分のCPI、PPI、URの値が入ります。

開始する前に、スタックをクリアし、数値表示形式をFIX2に切り替えましょう。

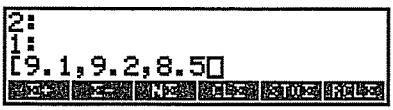
[CLEAR]
[MODE] 2 [FIX]

STATメニューのCLΣ(clear statistics、統計データのクリア)を使って、以前の統計データをクリアしましょう。(既存の変数ΣDAT を削除します。)

[STAT] [CLΣ]

1975年のデータをキー入力しましょう。

[9.1,9.2,8.5



このデータをΣDAT内に保存しましょう。

[Σ+]

ΣDATという名の新しい行列が自動的に作られました。1975年のデータがΣDATの1行目にあります。
1976年用のデータを入力しましょう。

[5.8,4.6,7.7 [Σ+]

1976年用のデータがΣDATに追加されて、統計用行列の2行目になりました。

1977年用のデータを入力しましょう。

[6.5,6.1,7 Σ+

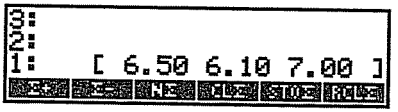
1977年用のデータがΣDAT に追加されて、統計用行列の3行目になりました。

データの修正

キー入力を間違えても、 Σ+ を押す前の間違いの訂正なら、コマンド行の修正だけでできます。しかし1976年のデータを入力した後にデータの間違いに気が付いたと仮定しましょう。その場合はそのデータをスタックに戻して間違えたデータを修正し、そのデータをΣDATに入れ直します。

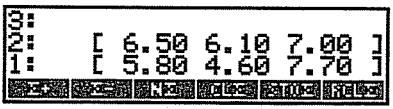
1977年のデータ(ΣDAT内の最後の行)を取り出して、スタックに戻しましょう。

Σ−



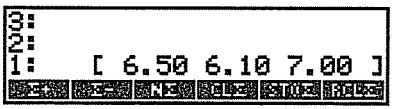
1976年のデータ(ΣDAT 内の最後の行)を取り出して、スタックに戻しましょう。

Σ−



このデータの中に間違いを見付けたら、 EDIT を押してデータをコマンド行に戻し、そのデータを修正し、次に ENTER を押して訂正したデータをスタックに入れ直します。(第18章の「既存のオブジェクトの修正」を見てください。)
1976年の修正したデータをΣDAT に入れます。

Σ+



1977年のデータをΣDATに入れます。

Σ+

さて残りのデータ(1978と1979年用)を入力して、5行分のデータが入っているかを調べましょう。

[7.6,7.8,6 Σ+
[11.5,19.3,5.8 Σ+
NΣ

1変量の統計

ここではCPI、PPI、URの平均、サンプルの標準偏差、不偏分散を求めます。CPI用のデータはΣDAT の1列目、PPI用のデータは2列目、UR用のデータは3列目にあります。
STATメニューの次の行に切り替えましょう。

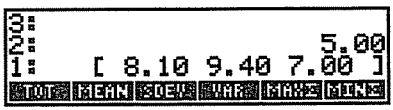
NEXT

ここに平均、サンプルの標準偏差、不偏分散のコマンドがあります。

平均の算出

平均を計算しましょう。

MEAN



CPIの平均は8.1、PPIの平均は9.4、URの平均は7です。

サンプルの標準偏差の算出

サンプルの標準偏差を計算しましょう。

SDEV

CPIのサンプルの標準偏差は2.27、PPIのは5.8、URのは1.14です。

不偏分散の算出

不偏分散を計算しましょう。

VAR

CPIの不偏分散は5.17、PPIのは33.64、URのは1.3です。

2変量統計

ここではCPIとPPIの相関係数とサンプルの共分散を求め、CPIとPPIの間に直線関係があるものとしてCPIの値からPPLの予想値を求めます。
STATメニューの3行目に切り替えましょう。

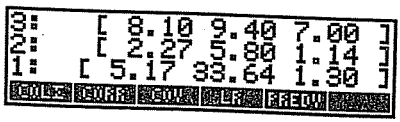
NEXT

ここに相関係数、共分散、直線回帰係数、予想値算出用のコマンドがあります。

使用する列の指定

２変数の統計を計算する前に、統計行列ΣDAT内のどの列に独立変数と従属変数のデータがあるかを指定します。この例では、１列目のCPIが独立変数に、２列目のPPIが従属変数になるようにします。
列１と２を独立変数と従属変数のデータとして指定しましょう。

1,2 [COL Σ]



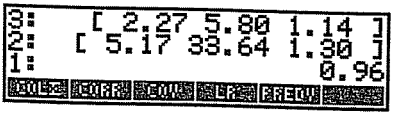
数値の１と２はΣPARという変数名のリスト内に保存されました。このΣPARは普通のリストですが、統計計算用に指定された名称を使っています。２変量統計のコマンドはどれも自動的にΣPARを参照します。

独立変数と従属変数のデータが入っている列を指定しないと、計算機は列１と２を使います。この例では列の指定は不要ですが、独立変数と従属変数のデータが列１と２に入っていないときには [COL Σ] を必ず実行してください。

相関係数の算出

相関係数を求めましょう。

[CORR]

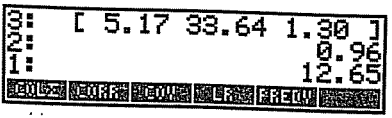


CPIとPPIの相関係数は0.96です。

サンプルの共分散

サンプルの共分散を求めましょう。

[COV]



CPIとPPIのサンプルの共分散は12.65 です。

直線回帰係数の算出

CPIとPPIのデータに一番よく当てはまる直線の係数を求めましょう。

LR

直線のy切片は−10.43 で、直線の傾きは2.45です。y切片と傾きはΣPARリストにも保存されています。

予想値の算出

CPIの値が 6 と 7 のときPPIの予想値を求めたいものと仮定します。この予想値はΣPAR内に保存しているy切片と傾きから計算できます。
CPIが 6 のときのPPI の値を予想しましょう。

6 PREDV

予想値は4.26です。
CPIが 7 のときのPPIの値を予想しましょう。

7 PREDV

予想値は6.71です。

第13章

ビット整数演算

この章ではビット整数の計算方法を説明します。各ビット整数はワード長が1～64ビットの範囲で符号なしの2進数を表します。2進数の入力と結果の読み取りが簡単になるように、基数を10進、16進、8進、2進に切り替えることができます。しかし、これを切り替えてもビット整数の内部表現には影響がなく、ビット整数に対するコマンドはビットごとに働きます。

BINARYメニュー内の全コマンドの要約は付録Dの「メニュー案内」にあります。詳細は、Reference Manualの "BINARY" を見てください。

ワード長の切り替え

ワード長はコマンドが返してくるビット整数の長さと、スタック内にあるビット整数の表示に影響します。ワード長は1～64ビットの範囲で変えることができ、この初期値は64ビットです。ここではワード長を16ビットにしたいと仮定しましょう。
例題を始める前に、スタックをクリアして、BINARYメニューに切り替えましょう。

[CLEAR] [BINARY]

ワード長として16ビットを指定しましょう。

16 STWS

これで16ビットよりも長いビット整数をキーから入力しても、最下位の16ビットだけが表示さます。

基数の切り替え

基数はスタックのビット整数の表示方法に影響します。基数は10進、16進、8進、2進のどれかに切り替えできて、初期値は10進です。
16進に切り替えてみましょう。

HEX

HEX の記号に小さな四角が付いて、基数がHEXであることを示しています。

ビット整数の入力

メモリ・アドレスの例として$24FF_{16}$を入力しましょう。（ビット整数の識別記号として # が必要です。）

24FF ENTER

右端の小文字のhは基数記号で、この時点で設定されている基数がHEX（16進）であることを表します。数値の入力のときには、数値がその時点の基数と違うとき以外は、基数記号のキー入力は不要です。
このビット整数が他の基数ではどんな表現になるか調べてみましょう。ビット整数そのものを変える必要はなく、基数の設定を変えるだけです。

基数をDEC（10進）に変えましょう。

DEC

基数をOCT（8連）に変えましょう。

OCT

基数をBIN（2進）に変えましょう。

BIN

基数をHEXに戻しましょう。

HEX

ビット整数の計算

このアドレスから$1FO_{16}$少ないアドレスを計算しましょう。

1FO −

他の数値計算の場合と同じように、差がスタックの1段目に返りました。

計算にビット整数と実数を混ぜて使うこともできます。普通の実数整数（識別記号の#を付けないで入力したもの）は、その時点の基数設定に関係なく、10進として解釈しています。

例として、このアドレスから27_{10}少ないアドレスを計算しましょう。

27 −

差は、2進整数として表現されて、スタックの1段目に返りました。

第14章

単位の換算

この章では単位の換算を説明します。この単位の換算とは、ある単位系での物理的測定量の数値を、他の単位系での数値に変換することです。詳細はReference Manualの〝UNITS〟を見てください。

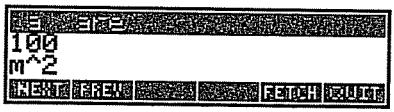
UNITSのカタログ機能

UNITSのカタログには、HP-28Sに組み込みの単位が約120個入っていて、ABC順に並んでいます。単位のつづりや定義を調べるときにこれを使います。まずスタックをクリアし、数値表示形式をSTDに切り替えましょう。

[ON] [CLEAR] [ON] [MODE] [STD]

UNITSカタログに入りましょう。

[ON] [UNITS]

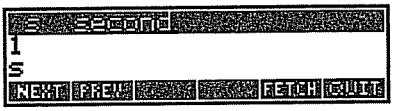


最初の単位はare（アール）で、単位記号はaです。これは面積の単位で100m²と同じことです。

メニュー・キーの [NEXT] または [PREV] を押し続けると、カタログ内容が前方または後方に順々に切り替わるので、試してみてください。

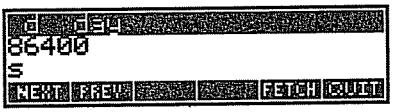
文字キーを押すと、その文字で始まる最初の単位に切り替えることができます。

[S]



これはsecond (秒) 用の見出し、単位記号がsで、値は 1 秒です。秒は基本単位なので、それ自体で定義されています。
HP-28Sを使うときはUNITSカタログ内にある単位記号を、実際にカタログで見える通りに使ってください。例えば、HP-28Sは小文字のsを秒として認識しますが、大文字のSは認識しません。
次にday（日）用の見出しを調べましょう。

[D]

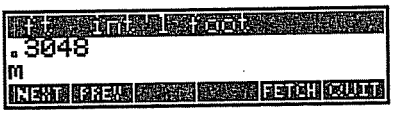


この見出しから、正しい単位記号はdで、値は86,400秒であることがわかります。
次にfoot（フィート、長さの単位）を探しましょう。

[F]

これはfrad(ファラド、静電容量の単位)の見出しです。前方に 7 個分進みましょう。

NEXT NEXT NEXT NEXT
NEXT NEXT NEXT



この表示からこれは国際フィートの見出しであることがわかります。このカタログにはフィートが 2 種類あって、次の単位は測量フィートです。
国際フィートの単位記号をコマンド行にコピーすることもできます。

FETCH

普通の表示に戻って、コマンド行に単位記号が見えます。
この章のこれからの操作例では、単位記号を直接キー入力していますが、場合に応じて [UNITS] と [FETCH] を使うこともできます。
コマンド行をクリアしましょう。

[ON]

単位の換算

最初に15℃を華氏に変換することにします。
スタックに数値を入れましょう。

15 [ENTER]

摂氏の単位記号を入力しましょう。

[ ] [°]C [ENTER]

単位記号が名称に変わりました。
華氏の単位記号を入力しましょう。

[ ] [°]F [ENTER]

この単位記号も名称に変わりました。
変換前の単位の数値を変換後の単位に換算しましょう。

[ ] [CONVERT]

この結果から15℃を換算すると59°Fになることがわかります。

次の例では、40インチからmmに換算します。今度は [CONVERT] で自動的にENTER を実行させてみます。
スタックをクリアし、数値を入力しましょう。

[CLEAR]
40 [ENTRE]

インチの単位記号を入力しましょう。

[LC] in [ENTER]

mmの単位記号をキー入力して、単位を換算しましょう。

UNITSカタログにはmmは見つかりません。これは接頭語の付いた単位記号で、メートルの単位記号mの前にミリ（1/1000）を表す接頭語のmがついたものです。同様に、kmはキロメーター用の接頭語付き単位記号、msはミリ秒用の接頭語付き単位記号です。接頭語の表はReference Manualの"UNITS"にあります。

[LC] mm [CONVERT]

この結果から40インチは1,016mmであることがわかります。

組立単位の換算

HP-28Sでは文字列とは文字が入っているオブジェクトです。文字列を利用してもっと複雑な単位を定義することができます。

組立単位は、"ft^2"のように単位の累乗を表したり、"ft*lb"のように単位の積、あるいは単位の累乗と積を組み合わせたものを表すものです。

組立単位では、"m/s" のように単位の商を表すこともできます。しかし、/記号は 1 個の組立単位の中に 2 回以上は使えません。/記号の前の分子側単位の全部と、/記号の後の分母側単位の全部を、それぞれグループ化してください。例えば、「フィート毎秒毎秒」は "ft/s^2" と表します。

次の例では、1 マイル毎時をフィート毎秒に換算します。
スタックをクリアして、数値を入力しましょう。

[CLEAR]
1 [ENTER]

マイル毎時の単位を入力しましょう。

[LC] mph [ENTER]

フィート毎秒の単位をキー入力しましょう。
フィート毎秒用の単位は組み込みでないので、組立単位を使います。

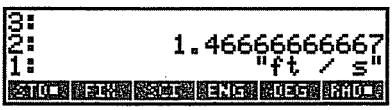
[ ] ["] [LC] ft [÷] s

文字列のキー入力を始めると、文字入力モードに変わります（カーソルの形が変わるのでわかります）。文字入力モードのときにコマンド行に書き込んだコマンドは、文字列を完成させるために [ENTER] を押すことが必要になります。

[ENTER]

元の単位の数値を新しい単位に換算しましょう。

[ ] [CONVERT]



この結果から 1 マイル毎時は1.46666666667フィート毎秒になります。

次は10立方フィートをガロンに換算しましょう。
スタックをクリアして、数値を入力しましょう。

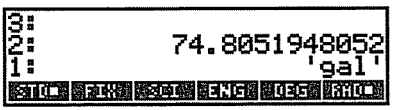
▒ CLEAR
10 ENTER

立方フィート用の組立単位を入力しましょう。

▒ " LC ft▒ ^ 3 ENTER

「米ガロン」用の単位を入力して、換算しましょう。

LC gal▒ CONVERT



この結果から10立方フィートは74.8051948052ガロンになります。

単位の点検

正しくない単位を使うと、期待通りでない結果の数値、または
`Inconsistent Units` エラーになってしまいます。この場合はUNITS カタログ、またはReference Manualの〝UNITS〟を調べてください。

期待通りでない結果の数値になったときは、次元は正しいけれども換算値が正しくない単位を使った可能性があります。例えば、1エーカー（acre、面積の単位）を〝ft^2〟に換算すると、結果は43,560より少し大きくなります。これはフィートには、〝ft〟（国際フィート）と、〝ftUS〟（測量フィート）の2種の単位があるからです。1エーカーを〝ftUS^2〟に換算すると正確に43,560になります。

`Inconsistent Units` エラーになるのは、正しくない次元の単位を使ったときです。例えば、力の単位としてlb（pound、ポンド）を使うとこうなります。力の正しい単位はlbf（pound-force、ポンド重）です。

単位換算用のユーザ定義関数

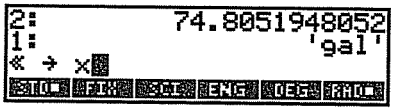
頻繁に特定の単位換算をするなら、その換算用のユーザ定義関数を作ることができます。ここではオンスとグラムとの間の換算用のユーザ定義関数のO→GとG→Oを作ります。どちらもユーザ定義関数なので、RPNと数式方式のどちらの計算方式にも使えます。

ユーザ定義関数として満たすことが必要な条件を思い出してみましょう。

- その引き数を明示することが必要です。
- 結果を正確に1個返すことが必要です。

最初にO→Gを作りましょう。
プログラムを始めて、引き数を示しましょう。

« ▒ → LC x

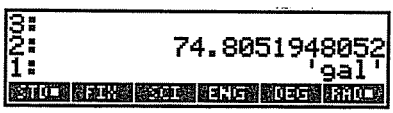


この右矢印で次の名称がローカル変数であることを示します。ローカル変数はこのプログラムの実行中にだけ存在します。
換算を定義しましょう。

« x ' oz ' ' g ' ▒ CONVERT
DROP ENTER

右側の認識記号が自動的に追加されました。
このプログラムの意味は次の通りです。スタックから（RPN方式のとき）または数式から（数式方式のとき）引き数を取り出して、それをxと名付け、xをオンスからグラムに換算し、スタックからグラムの単位を削除します。
このプログラムを変数O→Gに保存しましょう。

' O▒ → G STO



次にG→Oを作りましょう。
プログラムを開始して、引き数を示します。

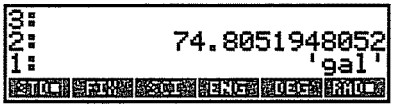
« → LC x

換算を定義しましょう。

« x ' g ' ' oz ' CONVERT DROP ENTER

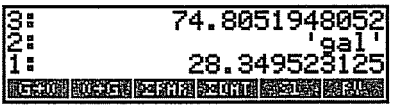
このプログラムの意味は次の通りです。スタックから（RPN方式のとき）または数式から（数式方式のとき）引き数を取り出して、それをxと名付け、xをグラムからオンスに換算し、スタックからオンスの単位を削除します。
このプログラムを変数G→Oに保存しましょう。

' G → O STO



換算を試験するために、１オンスが何グラムになるかを調べ、次にこの結果をオンスに戻すように換算しましょう。この結果は１になるはずです。
１オンスをグラムに換算しましょう。

1 USER G→C



１オンスは約28グラムです。次のこの結果をオンスに戻しましょう。

G→O

このように正確に逆換算できました。

第15章

プリンタの操作法

この章ではHP-28SとHP 82240Aプリンタを使うときの基本的なコマンドの一部を説明します。HP-28Sとプリンタとの位置関係と、プリンタのスイッチを入れる方法などは、プリンタの取扱説明書を見てください。

PRINTメニュー内の全コマンドの要約は付録Dの「メニュー案内」にあります。詳細は、Reference Manualの〝PRINT〟を見てください。

表示内容の印字

次のようにすると、表示画面の表示内容を印字することができます。

1. ON を押したままにします。
2. L （上側にPRINTと印刷してあるキー）を押します。
3. ON を放します。

このキー操作はコマンドPRLCD（print LCD、液晶画面の印字、PRINTメニューの2行目にあります）と同じ働きです。このキー操作を使うと、計算操作の途中でも、いつでも表示内容を印字することができます。

プログラム中で表示内容を印字したいときは、PRINTメニュー内にある、PRLCDコマンドを挿入するだけでできます。
スタックをクリアして、PRINTメニューに切り替えましょう。

CLEAR
PRINT

- PR1（print 1）はスタック1段目のオブジェクトを印字します。
- PRST(print stack)はスタック内にある全部のオブジェクトを印字します。
- PRVAR（print variable）は変数名とその内容を印字します。
- PRLCD（print LCD）は表示内容を印字します。
- CR（carriage return）は印字用紙を1行進めます。
- TRAC(trace on/off)はトレース印字モードに入ったり、抜け出たりします。

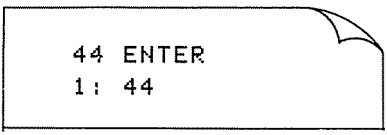
実行過程の記録

計算の進行過程を印字するには、トレース印字モードに入ります。

TRAC

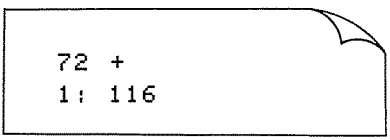
TRAC　メニュー記号に小さな白い四角が現れて、トレース印字モードに入ったことを表します。
ここでは2個の数値（例えば、44と72）を足して、どうなるか見てみましょう。まずスタックに44を入れましょう。

44 ENTER



入力したものとスタック1段目の結果を印字しました。
次に72を足しましょう。

72 +



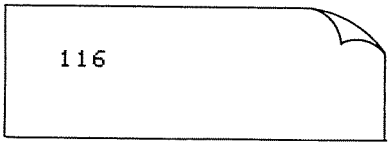
次に入力したものとスタック1段目の結果を印字しました。
トレース印字モードから抜け出ましょう。

TRAC

スタック1段目の印字

トレース印字モードに入って全部の結果を印字するよりも、PR1を使えば特定の結果だけを印字できます。

PRI

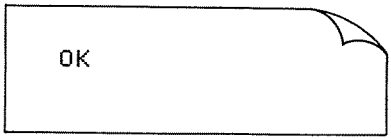


結果は1段目に残り、変化しません。
1段目に入れた文字列を印字することで、メモや注記などを印字することができます。OKというメッセージを印字するには、まずスタックに文字列を入れます。

" OK ENTER

次にメッセージを印字しましょう。

PRI

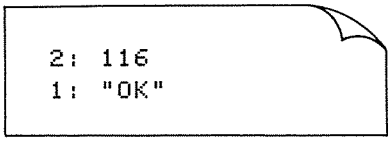


文字列の中身だけが印字され、引用符は印字されません。

スタック全体の印字

PRSTを使うと、スタック各段のオブジェクト全部が印字できます。

PRST



スタックの内容は変化していません。

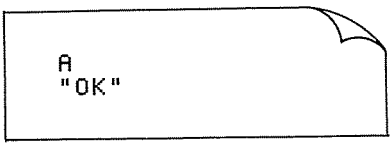
変数内容の印字

特定の変数の内容をスタックに呼び出さなくても、その変数名と変数の内容を印字することができます。これを試すために、文字列〝OK〟をAと名付けた変数に保存し、次に変数Aを印字してみましょう。
値が〝OK〟の変数Aを作りましょう。

' A STO

この変数名と内容を印字しましょう。

' A PRVAR



変数名はスタックから消えました。

第2部

基本的機能の要約

第16章

オブジェクト

この説明書の第１部の操作例にあるように、HP-28Sの基本的なオブジェクト型として９種類がありました。オブジェクトはこの計算機内の基本的な処理対象物で、これを作成することで問題を定式化し、これを操作することで、答えを求める処理ができます。

大部分のオブジェクト型の目的は、特定のデータ型を提供することでユーザの作業量を減らすためのものです。例えば、複数の実数を使って配列を表したりとか、配列内の各要素を個々に追い求めたり、あるいは配列どうしを算術演算するプログラムを書いたりするのは非常に大変です。それよりも配列オブジェクトに数値をまとめて入れる方が簡単で、こうすると１個の対象物として処理することができ、普通の算術用関数を使って計算の指示ができるようになります。

しかし、オブジェクト型が複数ある理由は単にデータの型を増やすためだけではありません。計算機の記号処理機能とプログラム機能は、記号表現のオブジェクト（名称と数式）とプログラム・オブジェクトという考えを基にしています。この両オブジェクトは単なるデータではなく、評価という働きかけをするとある結果を生み出すものです。（オブジェクトの評価は第23章で説明します。）

複数のデータ型、記号のままの処理、オブジェクトという考えに基いたプログラム作成などのおかげでHP-28Sを利用するうえで、覚えるべき規則は最小限になっています。例えばオブジェクト型に関係なく、どのオブジェクトでもまったく同じ方法でコマンド行にキー入力し、スタックに入れ、あるいは変数に保存できます。

この章はこれまで学習してきたことをまとめ、もう少し詳しい情報と、使用法のヒントなどを追加したものです。

実数

HP-28Sの実数は-10^{500}よりも大きくて、10^{500}よりも小さい数値です。この数値は内部的には仮数部（1～9.99999999999の範囲）、仮数部の正負符号、指数部（0～499の範囲）、そして指数部の正負符号として記憶されています。

HMS（時、分秒）形式の場合　時、分、秒（または度、分、秒）で表した数値の足し算と引き算にはHMS+とHMS−コマンドを使うことができます。足し算と引き算以外の計算には、まずHMS→を使ってHMS形式の数値を普通の10進小数に変換することが必要です。（詳細はReference　Manualの"TRIG"を見てください。）

複素数

複素数オブジェクトは、複素数の実数部と虚数部を表す2個の実数、または平面の1点を表す2個の実数を順序をつけて並べたものです。

直交座標と極座標　第7章と第8章で、プロットと座標読み取りに複素数を使いました。どの複素数も直交座標系（つまりx軸とy軸からの距離）で表していました。

第6章では極座標系（原点からの距離と角度で表す方法）を説明し、極座標と直交座標との間を変換するR→PとP→Rコマンドの使用法も取り上げました。座標のキー入力と結果の表示には極座標を使うこともできますが、計算には直交座標を使う必要があります。86ページのユーザ定義関数PSUMは、極座標の座標を直交座標に変換し、それを足して、極座標に戻すことで、極座標の足し算をする関数です。

数式オブジェクト内の場合　数式オブジェクト内に複素数をキー入力するには、`'SIN((0,1))'` のように、括弧を2重にすることが必要な場合もあります。外側の括弧は関数 `SIN( )` を表すのに必要で、内側の括弧は複素数の識別記号です。

ビット整数

ビット整数はビットが並んだものを表しています。並びの長さは、1～64ビットの範囲で、使用時点のワード長によって決まります。使用時点の基数でビット数をどう表すかが決まりますが、計算機内での内部表現には影響がありません。

大きな整数　基数を10にしてビット整数を使うと、19桁までの正の整数を正確に表すことができます。これは実数を使うより7桁多く正確に表すことができるということになります。

プログラム例　257ページの「ビット整数の表示」にあるプログラムは、ビット整数を4種の基数の全部で同時に表示します。

フラグ設定状態の保存　RCLF（recall flags、フラグの呼び出し）コマンドは64個のユーザ・フラグ全部の状態をビット整数の形で返します。STOF（store flags、フラグの保存）コマンドはビット整数の引き数の通りにユーザ・フラグを設定します。この両コマンドの使用例は、上記の「ビット整数の表示」のプログラムの一つである、"PRESERVE" で見られます。

文字列

文字列とは文字を並べたものです。第1部では文字列を次のように使いました。

- 第14章の「単位の換算」では組立単位を表すのに文字列を使いました。
- 第15章の「プリンタの操作法」では、メッセージの印字のために文字列を入力しました。DISPコマンドを使ってメッセージを画面に表示することもできますが、この方法は第27章の「対話型プログラム」で説明します。

文字列の一番大きな用途は文を表すことですが、個々の文字は0～255の範囲の数値で表すことができます。コマンドのCHR（character、数値の文字化）とNUM（character number、文字の数値化）で、文字とその数値との間の変換ができます。

キーボードにない文字 数値を入れてからCHRを実行することで、HP-28Sのキーボード上にない文字でも、表示することができます。印字はできても画面に表示できない文字もあります。全文字の表はReference Manualの"STRING"を見てください。

グラフィクス文字列 LCD→（LCD to string、表示画面を文字列に）コマンドは、それまで表示していた画面イメージを表す文字列を返します。この文字列をグラフィクス文字列と呼びます。→LCD（string to LCD、文字列を表示画面に）コマンドは、引き数の文字列が表す画面イメージを表示します。この両コマンドの詳細は、Reference Manualの"STRING"を見てください。

文字列操作 257ページの「ビット整数の表示」のプログラムでは、オブジェクトから文字列への変換、文字数の計数、2個の文字列の連結方法などが見られます。

配列

HP-28Sの配列は一次元（ベクトルと呼びます）または二次元（行列と呼びます）のどちらかで、その要素は実数または複素数です。第11章の「ベクトルと行列」で、配列の基本的な計算方法を取り上げました。そのほかに第1部には次のような配列の使用法もありました。

- 第11章では、要素がn個の定数ベクトルと、要素がn×n個の係数行列を使って、未知数がn個のn元連立一次方程式を解く方法も取り上げました。この処理法と精度については、Reference Manualの"ARRAY"を見てください。
- 第12章の「統計」では、統計計算用の指定変数ΣDATに統計データを行列として入れました。

数式内での関数的表現 配列を変数に保存すると、関数のように変数名を使うことで、配列の個々の要素を指定することができます。例えば、
'V(3)+V(5)'のようにベクトルVの3番目と5番目の要素の和を表すことができます。

配列の操作 262ページの「基本統計量」と、270ページの「統計データの中央値」に、各種の配列操作の例があります。

リスト

HP-28Sのリストとはオブジェクトを並べたのもです。一番多い使用法は、何個かのオブジェクトを1個のオブジェクトに組み合わせることです。第1部では次のようなリストの使用法がありました。

- 第4章の「計算を繰り返す方法」では、PATHコマンドが、HOMEディレクトリからその時点のディレクトリまでのディレクトリ名のリストを返しました。
- 第7章の「プロット」では、DRAWが使うパラメータが入ったリスト変数のPPARを取り上げました。
- 第8章の「SOLVE」では、初期推定値として、読み取った3点の座標値が入っているリストを使いました。
- 第10章の「微積分」では、積分する変数と、積分区間の下限と上限を指定したものを組み合わせて、リストにしたものを使いました。
- 第12章の「統計計算」では、2変量統計用のパラメータが入っているリスト変数のΣPARを使いました。

数式内での関数的表現　リストを変数に保存すると、関数のように変数名を使うことで、リスト内の個々の要素を指定することができます。例えば、'L(3)+L(5)' のようにリストLの3番目と5番目の要素の和を表すことができます。

リストとスタック　273ページのプログラムMEDIANには、リストの要素をスタックに入れる方法と、スタックにあるオブジェクトをリストに組み合わせる方法が見られます。

リストの並べ替え　270ページのSORTプログラムには、リストの要素を並べ替える方法が見られます。

リストからの要素の取り出し　272ページのLMEDプログラムには、リストから要素を取り出す方法が見られます。

名称

名称とは文字を並べたもので、他のオブジェクトを指定するために使います。名称には最長127文字まで使うことができますが、取扱い上の問題から、4または5文字以内にすることをお勧めいたします。

キーボードにある文字で名称に使えるのは、A～Zの英字、数字、それに記号 ? Σ π → μ " です。最初の文字は数字以外の文字に限ります。次の記号は名称に入れることができません。

- オブジェクトを分離するための記号。識別記号 (# ⌊ ⌋ " ' 〈 〉 ‹ › « »)、スペース、点、コンマ。
- 数式演算記号 (+ − * / ^ √ = < > ≤ ≥ ≠ ∂ ∫)。

HP-28Sはコマンド行を処理するときに、名称がクローバル用かまたはローカル用かを調べます。ローカル変数を作るプログラム構造で使っている名称は、そのプログラム構造内だけのローカル名です。これ以外の名称はグローバル名です。

ローカル名　第1部で作成したユーザ定義関数はローカル変数を作りました。この説明書では、グローバル用と区別しやすいように、ローカル用に小文字を使っていますが、コマンド→は、名称が小文字でなくても、ローカル名を作るということを覚えておいてください。ローカル変数にeまたはiと名付けると、そのローカル定義が組み込み定義よりも優先します。

グローバル名　第1部に現れていたこれ以外の名称はグローバル用です。例えば次の通りです。

- グローバル変数の名称(プロットまたはSOLVEで使った数値変数、USERメニュー内に見える変数全部)。
- ディレクトリの名称
- 特定の値を指定するのでなく、代数的に使った名称（代数計算、記号解、微積分)。

e、i、πを含めて、コマンド名をグローバル名に使うことはできません。それに、次の名称は特定の用途に予約済みです。

- EQはSOLVEとPLOTコマンドが解析に使う数式保存用の変数名です。
- ΣPARは統計コマンドが使うパラメータのリスト保存用の変数名です。
- PPARはプロット・コマンドが使うパラメータのリスト保存用の変数名です。
- ΣDATは組み込み機能で使う統計計算データの行列を保存する変数名です。
- s1、s2（以下同様）は、代数式のままの解の中で任意の符号を表すためにISOLまたはQUADが作り出すものです。
- n1、n2（以下同様）は，代数式のままの解の中で任意の整数を表すためにISOLが作り出すものです。
- 〝der〟で始まる名称はユーザ定義導関数を指します。

以上の名称のどれでもユーザが自由に使うことができますが、一部のコマンドが引き数として暗黙にこの名称を使うことを記憶しておいてください。

プログラム

プログラムはオブジェクトとコマンドを並べたものです。各プログラムは本質的にはコマンド行を1個のオブジェクトに入れたようなものです。コマンド行の内容をプログラム識別記号で囲むと、その内容を後で実行するために保存したいと示したことになります。

特別のプログラム・コマンドがPROGRAM BRANCH，PROGRAM CONTROL，PROGRAM TESTメニュー内にあります。このメニューの説明はReference Manualの中にあり、それと一緒に一般的な話題としての〝Programs〟中にもあります。

第1部では5個のプログラムを書き込みました。

- 第3章では変数を改名するプログラムを書き込んで、変数RENAMEに保存しました。
- 第5章ではコタンジェント（余接関数）用のプログラムを書き込んで、変数COTに保存しました。
- 第6章では極座標の足し算用のプログラムを書き込んで、変数PSUMに保存しました。
- 第14章ではオンスとグラム間の換算用のプログラムを書き込んで、変数O→GとG→Oに保存しました。

ユーザ定義関数　プログラムCOT、PSUM、O→G、G→Oはどれもユーザ定義関数です。どれもコマンド→と１個または複数の名称で始まり、この両方で１個または複数のローカル変数を定義し、その後に数式またはプログラムが続いているからです。ユーザ定義関数を変数に保存すると、組み込み関数のように、その変数名を数式内で使うことができます。

プログラム構造　コマンド→の後に名称と数式またはプログラムが続いたものを、ローカル変数構造と呼びますが、これはプログラム構造の一種として定義されています。分岐用のプログラム構造（IF...THEN...ELSE...ENDのような形など）や、ループ用のプログラム構造（DO...UNTIL...ENDのような形など）もあります。これについては、第26章の「プログラム構造」を見てください。同様に、第28章の「プログラム例」には20種のプログラムがあって、プログラム作成のこつと一緒に、プログラム構造の例があります。

無名のプログラム　変数に保存しておくことが不要なプログラムも便利です例えば、253ページの「数式を完全に展開して同類項をまとめる」や、257ページの「ビット整数の表示」を見てください。

数式

数式とは１個または複数の関数と関数の引き数を組み合わせたものです。この引き数には数値または名称、部分式が使えます。部分式とは数式を構成する要素です。数式は数式方式のつづり通りに書き込み、表示されます。この表記法は数学で一般的な表記法とよく似ています。数式は演算式と等式の２種に分けることができます。

演算式

第１部で使った演算式の使用法には、データとして、関数として、暗黙の等式（右辺に＝０があるものとみなすこと）として、の３種がありました。

データとしての演算式　２個の演算式を足す、演算式を二乗する、あるいは演算式を微分する、などのように演算式を計算した結果が別の式になることがあります。このような場合には、変数に代入した値とは関係なく、その演算式をデータの役目をはたすものとして処理してしまいます。

関数としての演算式　第4章で演算式RTOTを作りSOLVEを利用して、変数に数値を代入し、次にRTOTを評価して目的の結果を算出しました。この場合には演算式は、入力値を与えると、結果を導き出す関数としての働きをします。

暗黙の等式としての演算式　第8章で使ったSOLVEでは演算式のゼロ点を求めました。つまり、演算式の値が0になるような独立変数の数値を求めました。第9章ではQUADを使って代数式で表した根を求めました。つまり、元の演算式に値0を与えて、独立変数の代わりに、演算式を求めました。

どちらの場合も、演算式$f(x)$は等式$f(x)=0$のように働きました。

等式

等式は2個の演算式を等号（＝）で関係づけたものです。数学的には等号の使用法には2種類あります。

- 〝$x^2=4$〟や〝$x^2+y^2=1$〟のように、命題を示すため。この場合には等式の変数は限定された値だけを持つことができます。
- 〝$\sin 2x=2\sin x\cos x$〟や〝$y=3x^2+2x+5$〟のように、同等または定義を示すため。この場合には等式の変数には任意の値を与えられます。

HP-28Sでは、等式は命題的用法のためにだけ使います。〝$y=3x^2+2x+5$〟のような定義を実現するには、演算式'3*X^2+2*X+5'をYという名称の変数に保存します。

103ページの「均等払い複利計算」では、TVMとSPPVはどちらも数学的には等式として表現されます。等式として入力したTVM式の変数はある限られた値だけを持ちえます。SPPV式では、その値はその変数の値で定義されるので演算式を変数に記憶させます。

データとしての等式　2個の等式を足す、等式を二乗する、あるいは等式を微分する、などのように等式を計算した結果が別の等式になります。等式の両辺が別々に処理されます(各辺の演算式をデータとして処理します)。等式はその命題としての性質を継続するので、変数は限定された値だけを持つことができます。

等式を解く　103ページの「均等払い複利計算」でやったように、等式を数値的に解くときには、恒等性を満たす独立変数の代替物として等式を満たす演算式を求めます。

記号定数

数式には次の記号定数を入れることができます。どれも名称のように見えますが、実際には関数です。

- MINR（minimum real、最小の実数）は、HP-28Sで扱うことができる、最小の正の実数を表します。数値としての値は1.00000000000E-499です。
- MAXR（maximum real、最大の実数）は、HP-28Sで扱うことができる、最大の正の実数を表します。数値としての値は9.99999999999E499です。
- eは自然対数の底を表します。HP-28Sでの数値としての値は2.71828182846です。
- πは円周率を表します。HP-28Sでの数値としての値は3.14159265359です。
- iは虚数の$\sqrt{-1}$を表します。この値は（0，1）です。

数値定数モードまたは数値解答モードでは、記号定数を評価するとその数値が返ります。これ以外の時にはその記号のまま返ります。(定数モードの選択と解答モードの選択は第24章で説明します。)

第17章

操作、コマンド、関数

HP-28Sに組み込みの各手続きは、操作、コマンド、関数、解析可能関数に分類することができます。

- 操作とはHP-28Sに組み込んである手続き全般を言います。
- コマンドとはプログラムに入れることができる操作です。
- 関数とは数式の中に入れることができるコマンドです。
- 解析可能関数とはHP-28Sに導関数と逆関数が準備してある関数です。

組み込み手続きはたいていの場合最も大きく包括した機能で分類してあります。例えば、SWAPとIPはどちらもコマンドと言えますが、SWAPはコマンドとして分類し、IPは関数として分類します。下表は分類の例です。

操　　作

プログラムに入れることができない操作	コマンド		
	RPNコマンド	関数	
		解析可能でない	解析可能
INS	SWAP	ABS	ASIN
NEXT	DROP	∂	EXP
■ EDIT	LAST	IP	INV
■ VIEW↑	RCL	MAX	LN
ENTER	PURGE	OR	NEG
EEX	∫	%CH	SIN
■ COMMAND	STORE	R→D	SINH
■ UNDO	EVAL	R→P	SQ
■ CONT	CLEAR	XPON	+
ON	CONVERT	≠	=

Reference Manual末尾の "Operation Index" のType欄に、それぞれの組み込み手続きが、操作（O)、コマンド（C)、関数（F)、解析可能関数（A）のどれに当たるかの分類が入っています。およその目安として、各種類ごとに一般的なことを説明しておきます。

- プログラムに入れることができない操作の大部分は、キーを押したときにだけ実行できます。しかし、一部に操作と同等なものでプログラムに入れられるものが用意してあります。例えば、プログラム内で `21 MENU` を実行することは [TRIG] 操作（TRIGメニューへの切り替え操作）をラジアンに切り替える操作)をする効果があり、 `60 FS` を実行することは [RAD] 操作（角度単位）をする効果があります。
- RPNコマンドの大部分は、数学的な値の計算用ではなく、スタックの処理またはユーザ・メモリの変更用です。
- 解析可能でない関数の大部分は、逆算できない数学的計算をします。つまり、引き数がもつ特性の一部を返しますが、その結果から元の引き数に戻すことはできません。IP、FP、ABS、SIGNなどはこの例に入ります。
- 数学的には、定義領域内のどの点でもべき級数で表すことができる複素数の関数は解析可能な関数です。この場合にはその関数に逆関数と導関数があります。HP-28Sではこの定義に対して多少の例外があります。例えば、%関数には導関数がありませんが、微分可能であり、ABS関数には導関数がありますが、この関数は点0＋0iでは解析可能ではありません。

どの組み込み手続きもキー（キーボード直接、またはメニュー内）から実行することができます。キーを押したときの結果は、手続きの種別と入力モード（次の章で説明します）で変わります。

第18章

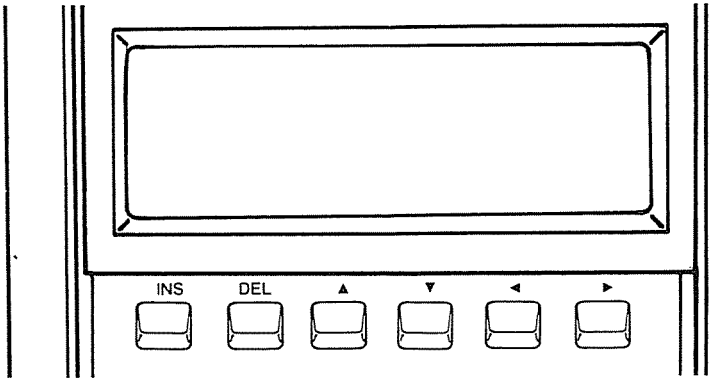
コマンド行

コマンド行にはオブジェクトを表すための文字をいくらでも入れることができます。オブジェクトのキー入力を始めるか、[EDIT]または[VISIT]を使って既存のオブジェクトの修正を始めると、コマンド行が表示部の最下行（メニュー記号があるときにはそのすぐ上の行）に現れます。

コマンド行には文字を１行以上入れられます。（左側キーボードのNEW LINEを使います。）もし１行に23文字以上を入れると、それまでの文字全体が左に移動して表示されます。左端の文字位置に、省略記号（...）が１個現れて、表示されていない文字があることを示します。カーソルを表示の左端よりも左に動かすと、表示されてなかった文字が現れ、文字全体が右側に移動します。そして、表示の右端が省略記号に変わります。入力内容が２行以上ある場合は、それらの行全部が左または右に移動します。

カーソル機能

カーソル機能は修正操作用の機能です。メニュー記号が画面に出ていないときにコマンド行上でこの機能が使えます。カーソル機能にはシフト・キーを併用する場合と併用しない場合の両方があります。シフト・キーを併用しないときの機能は、右ページの図のように、対応するメニュー・キーの上側に白で印刷してあります。



シフト・キーを併用しないカーソル機能のメニュー・キー（INSを除く）を押し続けると、キーを放すまでその働きを繰り返します。

キー	説　明
INS	重ね書きモードと挿入モードとの切り替え用です。重ね書きモードでは、入力した文字で既存の文字が書き替えられます。挿入状態では、カーソル位置の文字とその左側の文字との間に新しい文字が挿入されます。
DEL	カーソル位置の文字が削除されます。
▲	カーソルが1行上に動きます。
▼	カーソルが1行下に動きます。
◀	カーソルが1字分左に動きます。
▶	カーソルが1字分右に動きます。

シフト・キーを併用したときのカーソル機能（■ INSを除く）は、シフト・キーを併用しないでキーを押し続けたのと同じような効果があります。

キー	説　明
左シフト INS	カーソル位置よりも左側の文字全部が削除されます。
左シフト DEL	カーソル位置の文字を含めて右側の文字全部が削除されます。
左シフト ▲	カーソルがコマンド行の最上行に動きます。
左シフト ▼	カーソルがコマンド行の最下行に動きます。
左シフト ◀	カーソルがコマンド行の左端に動きます。
左シフト ▶	カーソルがコマンド行の右端に動きます。

その他の入力用キー

コマンド行にオブジェクトを入力するときに、次のキーが役に立ちます。

キー	説　明
◀▶	**カーソル機能とメニュー記号表示の切り替え用**　メニュー記号が画面に出ているときには、メニュー記号が消えてカーソル機能に切り替わります。カーソル機能になっていたときには、以前のメニューに戻ります。
CHS	**符号の変更**　カーソルが数値または正負符号の上にあるときには、その数値の符号が変わります。カーソルが数値または正負符号以外の部分にあると、負符号を書き込みます。(コマンド行が出ていないときには、NEGコマンドを実行します。)
EEX	**指数部の入力**　カーソルが指数部のない数値部分にあるときには、数値の後に文字Eを書き込みます。カーソルが指数部のある数値部分にあるときには、カーソルがEの右の位置に動きます。カーソルが数値以外の部分にあるときには、1Eを書き込みます。
◀	**1字取り消し**　カーソルの左側の文字を削除して、カーソル(それにカーソルの右側の文字全体)を1字分左に動かします。◀を押し続けると、このキーを放すまで同じことを繰り返します。
LC	**小文字**　大文字モードと小文字モードの切り替え用。コマンド行が画面に現れたばかりのときは大文字モードになっていて、A～Zを押すとA～Zを書き込みます。小文字モードではa～zを書き込みます。

[MENUS] **メニュー優先**　メニュー優先状態の切り替え用。メニュー優先状態になっているときには、左側のキーボードの上側3行([A]～[R])のどのキーも、シフト・キーを併用するときの働きと併用しないときの働きが逆になります。つまり[ARRAY]から[UNITS]までを押す前に、[MENUS]を押すことは不要になります。逆に文字AからRをキー入力するには、その前に[MENUS]を押すことが必要になります。

[ON]　**中断**　コマンド行の処理を中断して、コマンド行を取り消します。

オブジェクトの識別記号と分離記号

同じコマンド行に2個以上のオブジェクトまたはコマンドを入れるときには、次のどれかで互いに分離することが必要です。

- オブジェクトの識別記号。〈 〉[ ]{ } # " ' « ».
- 空白(スペース)または改行文字。[MENUS][NEWLINE]を押すと、コマンド行のカーソル位置に表示用ではない改行記号が入ります。コマンド行を実行したときには、改行記号は空白と同等の働きをします。
- コンマ(コンマを数値の整数部と小数部の分離記号に指定してないときに限ります)。

入力モードの区別

オブジェクトの入力が簡単になるように、入力モードを3種類（普通、数式、文字）が用意してあり、それぞれが異なる種類のオブジェクトに対応しています。先に進む前に、前の章の「操作、コマンド、関数」で区別したことを思い出してください。

- 操作はプログラム内や数式内に入れることができません。
- コマンドはプログラム内に入れることができますが、数式内には入れることができません。
- 関数（解析可能とそうでないものの両方）と名称はプログラム内や数式内に入れることができます。

コマンド行にオブジェクトを入力すると、計算機がこの区別を思い出します。操作キー（例えば、 ENTER ）を押すと必ずその操作を実行します。その時点の入力モードによって、コマンド・キー（例えば STO ）、または関数キー（例えば +）、USERメニューのキーを押したときにどうなるかが決まります。

普通入力モード このモードは数値、リスト、そして配列の入力用です。普通入力モードでは次のようになります。

- コマンド・キーを押すと、まず先にコマンド行を実行し、次にそのコマンドを実行します。
- 関数キーを押すと、まず先にコマンド行を実行し、次にその関数を実行します。
- USERメニューのキーを押すと、まず先にコマンド行を実行し、次に対応する名称を評価します。

数式入力モード このモードは名称と数式の入力用です。コマンド行を ' で始めると、自動的に数式入力モードになります。このモードでは次のようになります。

- コマンド・キーを押すと、まず先にコマンド行を実行し、次にそのコマンドを実行します。
- 関数キーを押すと、コマンド行に関数名を書き込みます。引き数が必要な関数なら、引数用に左括弧を自動的に加えます。
- USERメニューのキーを押すと、対応する名称をコマンド行に書き込みます。

文字入力モード このモードは文字列とプログラムの入力用です。 ▒ " または « を押すと、自動的に文字入力モードに入って案内表示に α が現れます。このモードでは次のようになります。

- コマンド・キーを押すと、コマンド行にコマンド名を書き込みます。
- 関数キーを押すと、コマンド行に関数名を書き込みます。
- USERメニューのキーを押すと、コマンド行に対応する名称を書き込みます。

カーソルがコマンド行の終端にあるか、挿入モードになっていると、それ以後のコマンドとの分離に必要な空白も自動的に入ります。

例外

普通入力モードまたは数式入力モードで、コマンド行を使っているときにでも、モードを切り替えることができるように、次のコマンド・キーを押すと、コマンド行を乱すことなくそのコマンドが実行できます。

- MODEメニュー内の　STD　、DEG　、RAD
- BINARYメニュー内の　DEC　、HEX　、OCT　、BIN

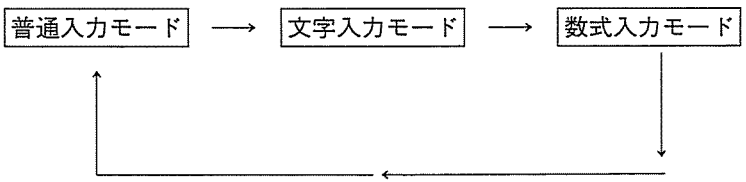
次のコマンドはプログラム内でだけ意味があるので、このキーのどれかを押すと、いつでもコマンド行にコマンド名を書き込みます。

- PROGRAM CONTROLメニュー内の　HALT
- PROGRAM BRANCHメニュー内の全部のキー

変数をうっかり削除しないように、 CLUSR （MEMORYメニュー内）を押してもコマンド行にCLUSRを書き込むだけです。このコマンドを実行するには ENTER を押すことが必要です。

入力モードの手動切り替え

名称または数式の前後に ' を押すたびに、普通入力モードと数式入力モードとの間で自動的に切り替わります。同様に [ ] " または « を押すと文字入力モードに切り替わります。 α を押すことで、入力モードを手動で切り替えることができます。この方法では入力モードが下図のように切り替わります。



入力モードの手動切り替え

これで [α] を1回か2回押すとどの入力モードにも切り替えることができます。[α] キーの使用法の例は次の通りです。

- 1回か2回だけ実行するプログラムを書くには次のように操作します。[α] を押して文字入力モードに切り替えます。プログラム識別記号を省略して、プログラムを入力します。[ENTER] を押してそのプログラムを実行します。[COMMAND] を押して、そのプログラムをコマンド行に戻します。[ENTER] を押してプログラムをまた実行します。
- 何個かの変数を一度に削除するには次のように操作します。[{] を押してリストを開始します。[α] を押して文字入力モードに切り替えます。削除したい変数に対応するUSERメニューのキーを押します。[ENTER] を押してこのリストをスタックに入れます。[PURGE] を押します。
- プログラムのキー入力中に、名称内に文字→を入れたいときには次のように操作します。文字入力モードになっているときに、[→] を押すと空白で囲まれたコマンド〝→〟を書き込んでしまうからです。[α] を押して数式入力モードに切り替えます。[→] を押します。[α] [α] と押して文字入力モードに戻します。

カーソルの形による入力モードの識別

カーソルの形でその時点の3種の入力モードと、挿入モードまたは重ね書きモードが識別できます。次表はその6種のカーソルを示したものです。

	挿入モード	重ね書きモード
普通入力モード		
数式入力モード		
文字入力モード		

コマンド行の実行

ENTER（またはその時点の入力モードでENTERを実行するキー）を押すと、計算機は次のように働きます。

1. 作業中であることを示す案内表示（(●)）が見えるようになります。
2. UNDOが可能状態に設定されているなら、この時点のスタックを別の記憶場所に保存します。
3. コマンド行内の文中にオブジェクトの識別記号と分離記号があるかを調べ、次に対応する文字列を部分文字列に分解します。
4. 文の部分文字列それぞれについて、文法通りであるかを調べて、オブジェクトの型を区別します。
5. COMMANDが可能状態に設定されているなら、コマンド行のコピーをコマンド・スタックに保存します。
6. コマンド行を実行します。
7. 作業中の案内表示（(●)）が消えます。

4番目の部分文字列の文法検査が不合格なら、5番目以降の作業はしません。その代わりに、Syntax Errorの表示を出し、文中の正しくない部分の最初の文字位置にカーソルを動かし、その部分を白抜き文字で強調します。エラー原因が文法的に未終了だったための場合は、カーソルが行の末尾に動きます。

既存のオブジェクトの修正

既存のオブジェクトをコマンド行に戻して内容を見たり、コマンド行を操作して修正したり、必要に応じて元のオブジェクトを修正したオブジェクトに一度で置き換えることができます。

キー	説　明
▒ EDIT	**スタックの1段目の修正**　1段目のオブジェクトがコマンド行に戻ります。
n▒ VISIT	**スタックのn段目の修正**　n段目のオブジェクトがコマンド行に戻ります。
'名称'▒ VISIT	**変数の修正**　指定した変数の内容がコマンド行に戻ります。

の操作でカーソル機能が働くようになり、文字入力モードになります。元オブジェクトが表示内に出ているときは白抜き文字に変わり、そのオブジクトの編集中でも、元のオブジェクトのコピーが残っていることを示します。

ブジェクトの点検または修正が終わったら、次のどちらかができます。

[ON]を押すと、修正を取り消して、コマンド行をクリアし、元のオブジェクトを修正しないで残します。
[ENTER]（またはENTERを実行するキー）を押すと、元のオブジェクトを書き換えます。

正が完了したときにカーソル機能が働いていると、上の操作で以前のメニーが表示に戻ります。

以前のコマンド行の回復

IP-28Sは直前に実行した4回分のコマンド行の内容を保存しています。[COMMAND]を1回押すと、最後に実行したコマンド行が返ります。[COMMAND]をもう1回押すと、その前回のコマンド行が返り、(コマンド行があるとそれを書き換える)以下同様です。[COMMAND]を5回以上押すと始めから繰り返します。

COMMANDの使用例は、47ページの「関数を間違えて実行したとき」と171ページの「入力モードの手動切り替え」にもあります。

MODEメニュー内の [CMD] を押すことで、この機能を止めることもできます。メニュー記号内の四角が消えて、コマンド行を保存しないことを示します。この機能が使えるようにするには、 [CMD] をもう一度押します。

文字列とコマンド行の関係

コマンド行にキー入力した文は文字列オブジェクト（つまり文字が続いたもの）の内容と同等です。プログラム的にコマンド行を実行するには、文字列内の文を入力してSTR→（string-to-objects、文字列のオブジェクト化）を実行します。プログラムを文の形で保存すれば、オブジェクトの形で保存するよりも、小さい容量ですむので便利です。STR→を実行したときに、コマンド行内のローカル名を識別します。

第19章

スタック

この章では、スタックについて学習したことを復習し、スタック上のオブジェクトを処理するコマンドを説明いたします。それにローカル変数の使用法とスタック処理を簡単にする方法も取り上げます。

スタックの概念の復習

スタックとは番号を付けた段が並んだもので、各段にオブジェクトを1個ずつ入れることができます。コマンド行にキー入力したオブジェクトは、ENTER を実行したときにスタックに入ります。コマンド行内の最初のオブジェクトが、スタックに最初に入るオブジェクトです。各オブジェクトはまず1段目に入り、それ以前のオブジェクトが1段ずつ上に動きます。スタックは（計算機のメモリの範囲内で）無制限に大きくなることができるので、以前に入れたオブジェクトの個数を考えながら入力する必要はありません。

一般に、コマンドはスタックから入力オブジェクト（これを引き数と呼びます）を取り出して、それを出力オブジェクト（これを結果と呼びます）に書き換えてスタックに入れます。例えば、＋関数は1段目と2段目から2個の引き数を取り去って、その合計を1段目に入れます。

コマンドを実行する前にその引き数がスタックにあることが必要です。引き数の後にコマンドが来るこのロジックを、スタック論理、後置論理、またはRPNと呼びます。RPNはポーランド出身の論理学者ヤン・ウカシェーヴィッチ（1878～1956）を記念して名付けた、逆ポーランド記法（Reverse Polish Notation）を略したものです。

コマンドの結果は次のコマンドの引き数として使えます。その結果をすぐに使わなくてもスタックにそのまま残るので、必要なときにはいつでも使えます。

スタックの1段目からオブジェクトを取り出すと、残りのオブジェクトは順に1段ずつ下がります。スタックにある不要なオブジェクトは捨てるか、変数内に保存するのが一番良い方法です。これで必要なスタックのオブジェクトの管理が簡単になります。同様に、新しい問題を始める前にスタック全体をクリアするのが一番良く、これでスタックにはその問題に関したものだけが残ります。

スタックを見る方法

いつもはスタックの始めの少しのオブジェクトだけが見えています。1段目のオブジェクトが大きいと、その最初の部分だけが見えます。▒ VIEW↑ と ▒ VIEW↓ の操作で、スタックのどこにあるオブジェクトでもその最初の部分を見ることができます。

この操作はスタックを見るための「窓」を動かすようなものです。この窓の大きさは1～4行の範囲で、メニューの有無とコマンド行の有無によって決まります。

キー	説　明
▒ VIEW↑	窓を上に（スタックの上段の方に）動かす。
▒ VIEW↓	窓を下に（最終的には1段目まで）動かす。

窓を動かしても、スタックの内容、コマンド行の内容、コマンドの働きには影響がありません。計算操作などの後で表示は元に戻ります。ON キーで中止することもできます。

スタックの操作

第1部ではスタック操作用の基本的コマンドの一部を使いました。それはCLEAR(スタックをクリアする)、DROP(1段目のオブジェクトを捨てる)、SWAP(1段目と2段目のオブジェクトの順序を入れ換える)でした。ここではスタックのオブジェクトの移動、コピー、削除用のコマンド全部の要点をまとめておきます。詳細はReference Manualの"STACK"を見てください。

スタック上のオブジェクトの移動　次のコマンドはスタック上のオブジェクトを並べ替えますが、オブジェクトの個数は変わりません。*n*が前に付いているコマンドは、実数で整数の引き数が必要です。

コマンド	説　明
SWAP	2段目のオブジェクトを1段目に動かす。
ROT	3段目のオブジェクトを1段目に動かす。
n ROLL	*n*段目のオブジェクトを1段目に動かす。
n ROLLD	1段目のオブジェクトを*n*段目に動かす。

コマンド名のROT (rotate)、ROLL、ROLLD (roll down) はオブジェクトのまとまった移動を表したものです。ROTは3段目のオブジェクトを1段目の動かすので、3個のまとまったオブジェクトをずらして移動させることになります。ROLLとROLLDは*n*個のオブジェクトをまとめてずらします。

スタック上のオブジェクトのコピー　次のコマンドはスタック上の1個または何個かのオブジェクトのコピーを返します。1個のオブジェクトをコピーすると、コピーしたものが1段目に入り、スタックのそれ以外のオブジェクト（コピー源のオブジェクトを含む）が1段ずつ順に上へ動きます。2個以上のオブジェクトのコピーのときには、コピーしたものが1群となり同じように処理されます。*n*が前に付いたコマンドは、実数で整数の引き数を必要とします。

コマンド	説　明
DUP	1段目のオブジェクトをコピーする。(コマンド行が出ていないときには、ENTERを押すことでDUPが実行できます。)
OVER	2段目のオブジェクトをコピーする。
n PICK	*n*段目のオブジェクトをコピーする。
DUP2	1段目と2段目のオブジェクトをコピーする。
n DUPN	1段目から*n*段目のオブジェクトをコピーする。

スタック上のオブジェクトの削除　次のコマンドはスタックから1個または複数のオブジェクトを削除します。残りのオブジェクトは順々に下に動きます。*n*が前に付いたコマンドは、正の整数で実数の引き数を必要とします。

コマンド	説　明
DROP	1段目のオブジェクトを捨てる。
DROP2	1段目と2段目のオブジェクトを捨てる。
n DROPN	1段目から*n*段目のオブジェクトを捨てる。
CLEAR	全部のオブジェクトを捨てる

ローカル変数

第1部でユーザ定義関数を少し説明しました。ユーザ定義関数とは、ローカル変数を定義して、その変数を数式内またはプログラム内で使うプログラムです。ユーザ定義関数は、組み込み関数のように数式内に入れることができます。

ローカル変数を使うと、スタック操作の必要性が減ります。ローカル変数を作ったときに、その値がスタックから取り込まれます。それ以後はスタックからその値を探し出す代わりに、名称でそれを参照することができます。

ローカル変数はユーザ定義関数だけでなくプログラム内でも使えます。第28章の「プログラム例」のかなりのプログラムがローカル変数を使っています。特に役に立つのは、241ページの「BOX関数」、253ページの「連続実行」、258ページの「以前のフラグ状態を保存して元に戻す」、270ページの「リストの並べ換え」です。

直前の引き数の復活

HP-28Sは最後に実行したコマンドの引き数を保存できます。コマンドによって、保存するオブジェクトが1～3個になります。(引き数を使わないコマンドを実行したときは、その前に保存した引き数をそのまま保存します。)LASTコマンドは保存している引き数を、前に占有していた通りのスタックに返します。

2個以上のコマンドで引き数が順番通りに正確に同じものが必要なときには、次のコマンド用にLASTを実行して引き数のコピーをスタックに戻すことができます。しかしコマンドが正確に同じ引き数を必要としないとき、またはコマンドが順番通りでないときには、ローカル変数を使うほうがやさしくなります。

MODEメニューの [LAST] を押してLAST（つまり、引き数の保存）機能を止めることもできます。メニュー記号から小さな四角が消えて、引き数を保存しないことを示します。エラーが発生したときには保存した引き数を使うので、このようにすることはお勧めできません。しかし、メモリ不足でコマンドまたはプログラムが実行できないときには、LASTが働かないようにすると実行できることがあります。終了後に、もう一度 [LAST] を押してLASTが働くようにしておいてください。

スタックの回復

[ENTER]（またはENTERを実行するキー）を押すたびに、HP-28Sはまずスタックのコピーを保存し、次に指定した働きを実行します。結果に満足できなかったら、■ [UNDO] を押すことで保存したスタックを元に戻すことができます。UNDOはスタックに影響を与えるだけで、ユーザ変数やユーザ・フラグは変化しません。■ [UNDO] の使用例は、47ページの「関数を間違えて実行したとき」を見てください。

MODEメニューの [UNDO] を押すことで、この機能を止めることができます。メニュー記号から小さな四角が消えて、スタックを保存しないことを示します。この機能が働くようにするには、もう一度 [UNDO] を押します。

リストとスタックの関係

スタック上の何個かの内容は1個のリストの内容と同等です。つまり、どちらも何個かのオブジェクトが並んだものです。DEPTH　→LISTを実行することで、スタックのオブジェクト全部を1個のリストに入れることができます。DEPTHコマンドはスタックのオブジェクトの個数を返し、→LIST(stack to list、スタックからリストに)コマンドは指定した個数のオブジェクトを1個のリストにまとめます。

LIST→コマンドでリストを開いてスタックに入れることが必要なことがよくあります。スタックで要素を処理した後に、→LISTコマンドでリストに再編成することもできます。この両コマンドの使用例は、273ページの「統計データの中央値」を見てください。

第20章

メモリ

HP-28S内では、コマンド行、スタック、回復機能、システム動作などの各種目的にメモリを使っています。コマンド行とスタックは第18～19章で説明しました。この章では主にユーザ用メモリ（ディレクトリを含む）を説明します。それにメモリ残量が少ないときのことと、それによる回復機能、本体操作への影響も説明します。

ユーザ用メモリ

ユーザ用メモリには変数を入れることができ、変数を組織的に管理できるようにディレクトリを作ることができます。

グローバル変数

変数とは名称オブジェクトとそれ以外のオブジェクトを組み合わせたものです。名称オブジェクトは変数名を表し、それ以外のオブジェクトは変数の値（変数の内容とも言います）です。

グローバル変数はユーザ用メモリ内に保存する変数です。ローカル変数もありますが、こちらはプログラム構文上で作り出して、そのプログラム構造の実行中にだけ保存する変数です。ローカル変数の主な用途はスタック操作の代用で、これは第19章の「スタック」で説明した通りです。この章で「変数」とはグローバル変数を指します。

どのオブジェクト型でも変数に保存できます。第１部では数値変数、プログラム変数、数式変数、リスト変数、配列変数を作りました。名称変数も作ることができ、この変数の内容は別の変数名になります。

変数の作成、呼び出し、削除には次のコマンドを使います。このコマンドは変数の内容に関係なく、どの変数でも処理できます。

コマンド	説　明
STO	指定した値と名称で変数を作ります。
RCL	指定した名称の変数の内容を呼び出します。
PURGE	指定した1個または2個以上の名称の変数を削除します。

ディレクトリ

第4章の「計算を繰り返す方法」で、2種類の直並列回路の合成抵抗を求めるためにSOLVEを使いました。そして2組の抵抗値はどちらの回路にも使えました。そこで学んだことをここで復習してみます。

ディレクトリ作成の主な動機は次の2点です。

- 特定の用途または目的用の変数をグループ化するため　電気工学の問題を解くためにEEディレクトリを作り、そこで問題を解くときに、関連した変数にだけ注目することができました。もうひとつ重要なことは、別の問題を解くときには、電気工学用の変数はEEディレクトリ内に隠れてしまうことです。
- 同じ名称を使った変数のグループを分離するため　EE内で変数R1、R2、R3に違う値を保存するためにSP1とSP2ディレクトリを作りました。ディレクトリを切り替えるだけで、値の組み合わせを一方から他方に切り替えることができます。

ディレクトリの作成　ディレクトリを作るには名称を入力して、CRDIR（create directory、ディレクトリの作成）を実行します。ディレクトリ名がUSERメニュー内に見えるようになります。新しいディレクトリを子ディレクトリ（または下位ディレクトリ）と呼び、そのディレクトリが入っているディレクトリを親ディレクトリと呼びます。

その時点のディレクトリ 最初に存在しているディレクトリは組み込み済みであるHOMEディレクトリだけです。別のディレクトリを作ると、どれをその時点のディレクトリにするかを選ぶことができます。言い換えると、USERメニュー内に見える変数の組み合わせを選ぶことができます。

その時点のディレクトリを切り替えるには、希望するディレクトリの名称を評価します。例えば、ディレクトリを作ったばかりなら、USERメニュー内の対応するキーを押すとそれがその時点のディレクトリになります。

ディレクトリを複数にする目的は利用できる変数を限定するためなので、変数を使う大部分のコマンドはその時点のディレクトリ内だけで働きます。その時点のディレクトリ内の変数だけを変えることができます。

MEMORYメニュー内の次のコマンドはその時点のディレクトリ内だけで働きます。

コマンド	説　明
VARS	その時点のディレクトリ内の変数名とディレクトリ名の全部のリストを返します。
ORDER	その時点のディレクトリ内の変数とディレクトリをリストで指定した通りに並べ替えます。
CLUSR	その時点のディレクトリ内の変数全部と、空のディレクトリ全部を削除します。

その時点のディレクトリ経路 PATH（経路）コマンドを実行するとどのディレクトリ内にいるかを調べることができます。このコマンドはHOMEディレクトリからその時点のディレクトリまでのディレクトリの順番を調べたリストを返します。

操作の実行中に、場合によっては計算機がその時点のディレクトリだけでなく、ディレクトリ経路全体を探索することがあります。この探索はその時点のディレクトリから始まります。変数が見つからないと、探索は親ディレクトリに移ります。この探索はHOMEディレクトリに戻るまで続きます。

名称の評価のときにもこうなります。結局、親ディレクトリの名称の評価がうまくできないと、親ディレクトリには戻れません。一重引用符で囲んでない名称をキー入力したときや、プロットさせたり SOLVER を使ったとき、スタック上の数式を評価したときなどにも名称の評価が起こります。

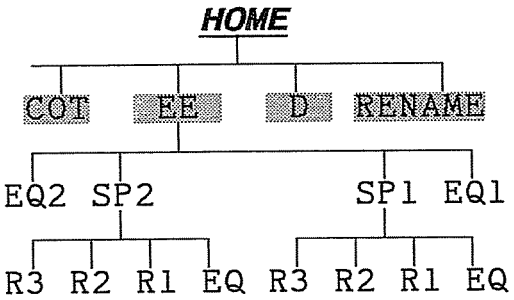
その時点の経路上を探索するこれ以外のコマンドにはRCLとPRVAR（print variables、変数の印字）があります。いずれにしてもその時点の経路を探すという動作はできても変数を変えることはありません。

HOMEディレクトリはいつもその時点の経路上にあるので、計算機はいかなる場合でもHOMEディレクトリ内の変数を探すことができます。そこでHOMEディレクトリ内の変数の一部を子ディレクトリに移すことで、いつも利用できる変数を制限することができます。

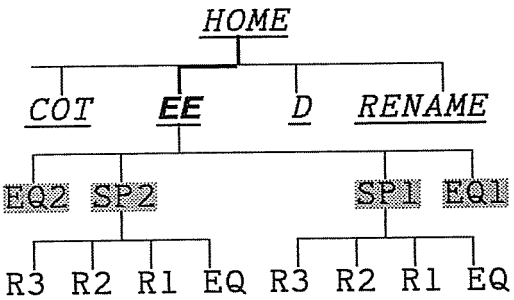
ディレクトリ構造　次ページの図は第4章で作ったディレクトリ構造を示したものです。最初の図は、HOMEがその時点のディレクトリです。2番目の図は、EEがその時点のディレクトリです。3番目の図は、SP2がその時点のディレクトリです。どの図も次の記号を使っています。

ディレクトリ構造図（次ページ）に使っている記号

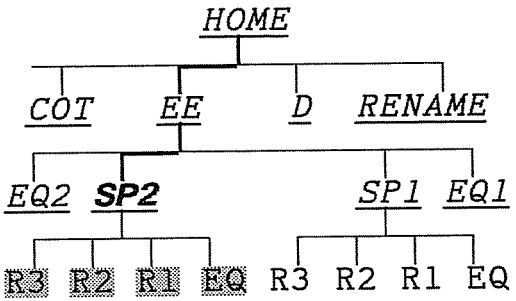
太字（**HOME**、**EE**など）	その時点のディレクトリ名。
網がけ（ 変数名 、 SP2 など）	その時点でUSERメニューに現れる名称。これらの変数の内容は書き換え可能です。
——	ディレクトリ経路。
斜体下線（*D*、*EQ1*など）	ディレクトリ経路上の名称。これらは評価したり、RCLやPRVARSが実行できます。しかし変数の内容の変更はできません。



HOMEディレクトリにいる時



EEディレクトリにいる時



SP2 ディレクトリにいる時

ディレクトリの削除　ディレクトリがからであれば変数と同様に削除することができます。削除したいディレクトリが入っている親ディレクトリに切り替え、スタックにディレクトリ名を入れ、PURGEを実行します。

削除したいディレクトリに変数や子ディレクトリが入っていたら、そのディレクトリを削除する前に、変数や子ディレクトリの削除が必要です。一般的な手順は次のようになります。

1. 削除したいディレクトリに切り替えます。
2. CLUSRを実行して、ディレクトリ内をからにします。
3. その親ディレクトリに移ります。
4. 目的のディレクトリを削除します。

手順2で Non-Empty Directory エラーになったら、そのディレクトリ内の子ディレクトリがからでないためです。この場合は手順1～3でその子ディレクトリをからにします。それから手順2～4を続けて、そのディレクトリを削除します。

ディレクトリ構造の上昇と下降　第28章にはディレクトリ構造上を上方に移動する（親ディレクトリのどれかに動く）または下方に移動する（子ディレクトリのどれかに動く）プログラムがあります。275ページの「ディレクトリの移動」を見てください。

回復機能

HP-28Sはコマンド行、引き数、そしてスタックのコピーを自動的に保存しています。このコピーで誤操作の回復ができます。つまり誤操作する前の状態に戻ることができます。これで計算開始時点に戻らないでも、計算のやり直しができます。コマンド行と引き数のコピーは繰り返し計算にも便利です。

このコピーがメモリのかなりの量を使うことがあります。この回復機能（コマンド行、スタック、引き数）のそれぞれごとに、この機能を使えるようにするか停止させるかを切り替えることができます。この回復機能の切り替えと表示はMODEメニュー内にあります。

一般的には回復機能のどれもが使えるようにしておくのがベストです。しかし、使用可能なメモリが少くて、回復機能が大きなオブジェクトを保存しているようなときには、個々の回復機能を使えなくすることによって保存したオブジェクトをクリアしないでも、使えるメモリを増やすこともできます。

メモリ不足

HP-28Sには32Kバイトのユーザ用メモリがあります。その内の約400バイトは計算機そのものが使う分です。実質的にはHP-28Sのどの操作でも（例えば、コマンドの解釈などでさえ)、いくらかのメモリを使います。一部の数式用のコマンド（COLCT、EXPAN、TAYLR）の使用メモリ量は、その引き数が複雑になるほど、急速に増加します。このような場合には数キロバイトのメモリを計算機の作業用に残しておくことが必要なことがあります。

MEMORYメニュー内にあるMEMを実行することで、使用可能な残りメモリ量を調べることができます。

HP-28Sのメモリはユーザのオブジェクトにも、HP-28Sの動作の作業用にも使うので、メモリにオブジェクトを詰め込み過ぎると、普通の計算機操作が困難になったり、できなくなったりします。HP-28Sにはメモリ不足の程度によって、次のように6段階の警告と応答が用意してあり、後のものほど厳しい状況です。

メモリが不十分 コマンドの実行に使えるメモリが十分でないと、`Insufficient Memory` エラーが発生します。LASTが使えるときには、元の引き数がスタックに戻ります。LASTが止めてあるときには、引き数が消えます。

UNDOの余裕がない スタックのコピーの保存に使えるメモリが十分でないと、`No Room for UNDO` エラーが発生します。UNDO機能は自動的に使えなくなります。UNDOを使えるようにするには、MODEメニュー内の UNDO を押します。

ENTERの余裕がない コマンド行を処理するのに使えるメモリが十分でないと、計算機はコマンド行をクリアして、`No Room to ENTER` を表示します。コマンド・スタックが使えるときには、失敗したコマンド行のコピーがコマンド・スタック内に保存されています。

EDITまたはVISITを使って、既存のオブジェクトを修正しようとしていてコマンド・スタックにENTERに失敗したコマンド行のコピーが保存されていたら次のように操作します。オブジェクトの元のコピーを削除します。[COMMAND]を押して、修正したオブジェクトが入っているコマンド行を回復します。[ENTER]を押して修正したオブジェクトを入力します。

メモリ不足　自由に使えるメモリが128バイト未満になると、表示の最上行にLow Memory!が1回点滅します。この表示は、使えるメモリがもっと多くなるまで、キー操作ごとに点滅します。計算を続ける前に、メモリ内の不要なオブジェクトを削除してください。

スタックを表示する余裕がない　ときによっては、仕掛かり中の操作が完了する途中で、普通のスタック表示に必要な自由に使えるメモリが不足することがあります。この場合は、表示の最上行にNo Room to Show Stackが現れます。普通はスタックのオブジェクトを表示する行には、そのオブジェクトの型だけの表示になります。例えば、Real Number, Algebraic, などです。

スタックのオブジェクトを表示するのに必要なメモリ量は、オブジェクトの型によって変わり、普通では数式が一番多くのメモリが必要です。メモリから1個または2個以上のオブジェクトをクリアしたり、スタックのオブジェクトを変数に保存すると、表示できなかったスタックが見えるようになります。

メモリがない　メモリ不足が極端な場合には、なにもできないほど（例えばスタックを表示する、メニュー記号を表示する、コマンド行を作る、など）になります。このような場合には、続行する前にいくらかのメモリをクリアする必要があります。特別のOut of Memory手順が働いて、次のような表示になります。

計算機は次の順番にクリアするかどうかを質問します。

1. COMMANDスタック（使えるとき）
2. UNDOスタック（使えるとき）
3. LAST引き数（使えるとき）
4. ユーザ定義メニュー（あるとき）
5. スタック
6. HOMEディレクトリ内の個々の変数

各項目を削除したいときはメニュー・キーの YES を押し、残しておきたいときは、 NO を押します。

少くとも1回は YES を押した後に、 ON を押すとOut of Memory処理の打ち切りを試すことができます。十分なメモリが使用できると、計算機は普通の表示に戻ります。メモリがまだ不足なら、警告音が出て削除手順の続行になります。上記1～6を一回りすると、Out of Memory処理は普通の操作に戻ろうとします。それでもまだメモリが不足していると、削除手順の最初の部分に戻ります。

からのディレクトリの削除のときに YES を押すと、そのディレクトリを削除します。変数が入っているディレクトリの削除に YES を押すと、そのディレクトリ内の変数名の表示になります。

効率の最大化

計算機はメモリを上手に使うために、時々メモリの管理をしていますが普通は気になることはありません。例えば、プロットの途中で一時的に休止するので、この処理に気がつく位です。しかし、メモリがほとんどいっぱいで、スタックに何百ものオブジェクトが入っていると、メニューの切り替えのような簡単な操作でも、計算機の応答が遅くなります。

ここでは管理量を減らすことで速度を最大にするのと、管理の効率を上げて使用できるメモリを最大にするためのヒントを記します。

速度を最大にするには次のようにします。

- スタックに何百ものオブジェクトを入れない。
- スタックに大きなリスト（何百ものオブジェクトが入っているような）を残しておかない。変数内に保存する。

使用可能なメモリ量を最大にするには次のようにします。

注記 次の手順で次の全部が消えてしまいます。スタック、回復用データ（COMMAND、UNDO、LAST）、その時点のユーザ定義メニュー（CUSTUM）、停止中のプログラム。

1. ユーザ用メモリ内の不要な変数とディレクトリを削除します。
2. 保存しておきたいスタックのオブジェクトを変数に保存します。
3. システム停止（ON ▲）を実行します。

これでディレクトリはHOMEになります。

配列計算に使うメモリ量を最小にするには次のようにします。 配列を変数に保存して、名称で参照できるようにします。スタックに入れたままで使うのを避けます。こうするための要点は次の通りです。

1. 中間結果用のも含めて、何個の変数が必要かを考えます。
2. 正しい型（実数または複素数、ベクトルまたは行列）の小さな配列を作ります。それを変数に保存します。RDMを使ってその大きさに合わせます。
3. STOREメニューにある変数演算コマンドを使って計算を実行します。
4. 個々の要素を処理するには、変数名と共にGET、GETI、PUT、PUTIを使うか、'A(5,6)' EVAL や 'B(3)' STOのような数式表現を使います。配列全体をスタックに戻すことはしないでください。

第21章

メニュー

HP-28Sのどの操作、コマンド、関数もキーボードまたはメニュー内にあります。メニューに切り替えると、表示部の最下行に6個以内のメニュー記号が現れます。この6個の記号の並びをメニュー行と呼び、それぞれがキーボードの最上行のメニュー・キーの定義を表します。(メニュー・キーがカーソル機能になるときはキーの上側に白で機能が印刷してあります。)

特定のメニュー切り替えキー(例えば▒ ARRAY または TRIG など)以外に、次のキーでメニューを操作できます。

キー	説　明
◀▶	**カーソル機能の切り替え**　カーソル機能が働いてないときには、このキーを押すとカーソル機能に切り替わります。カーソル機能が働いているときには、以前のメニューに切り替わります。
CUSTOM	**最後に作ったユーザ定義メニュー**　MENUコマンドで作った一番新しいユーザ定義メニューに切り替わります。
NEXT	**次のメニュー行**　メニュー記号の次の行に切り替わります。最後の行を表示していたときには、最初の行に切り替わります。
▒ PREV	**前のメニュー行**　メニュー記号の前の行に切り替わります。最初の行を表示していたときには、最後の行に切り替わります。
▒ MENUS	**メニュー優先**　メニュー優先にしたり、抜け出します。メニュー優先にすると、左側のキーボードの上3行のキー(文字キーの A ～ R)の▒キーを押したときと押さないときが切り替わります。メニュー優先のときには、A を押すとARRAYメニューに切り替わり、▒ A を押すと文字Aを書き込みます。

コマンドのメニュー

次の各メニューに組み込み機能用のキー記号が入っていて、大部分はプログラムに入れることができるコマンドです。各メニュー内のコマンドの要点は、付録Dの「メニュー案内」を見てください。Reference Manualには、各メニューをアルファベット順に並べてあり、それぞれの機能の説明があります。

このメニュー内のキーの働きは、169ページで説明した入力モードによって決まります。

メニュー	説　明
ALGEBRA	数式処理用のコマンド類。
ARRAY	ベクトルと行列用のコマンド類。
BINARY	ビット整数、基数変換、ビット操作用のコマンド類。
COMPLEX	複素数用のコマンド類。
LIST	リスト操作関係のコマンド類。
LOGS	対数、指数、双曲線関数。
MEMORY	ユーザ用メモリ、ディレクトリ操作。
MODE	表示モード、角度単位、回復機能の切り替え。
PLOT	プロット用のコマンド。
PRINT	印字用のコマンド。
PROGRAM BRANCH	プログラムの分岐構造作成用。
PROGRAM CONTROL	プログラムの制御、停止、シングル・ステップ操作。
PROGRAM TEST	フラグ、論理の条件判断など。
REAL	実数用のコマンド類。
SOLVE	数値と記号解用のコマンド、SOLVER。

メニュー	説　明
STACK	スタック処理。
STAT	統計と確立のコマンド類。
STRING	文字列。
TRIG	三角関数、座標と角度の変換。

操作のメニュー

次のメニューはプログラムに入れることができない操作です。

メニュー	説　明
カーソル機能	コマンド行の編集用。第18章に説明があります。
CATALOG	コマンドのカタログで、USAGE（使用法）の下位メニューもあります。第22章で説明します。
UNITS	換算に使える単位。第14章に説明があります。

変数のメニュー

メニュー	説　明
SOLVER	解析用に指定した式内の変数の保存と変数を解く。独特な表示（白地に黒文字）は独特な働きを表します。
USER	その時点のディレクトリ内の変数と子ディレクトリの表示。各キーの働きは、169ページで説明した入力モードによって決まります。

ユーザ定義メニュー

コマンドのMENUで、名称とコマンドを入れたリストを使うことによりユーザ定義メニューを作ることができます。このユーザ定義メニューはSOLVERメニューまたはUSERメニューに似たものです。

- リスト内の最初の要素をSTOコマンドにして、その後に名称をいくつか並べると、MENUでユーザ定義の入力メニューが作れます。このメニューの表示と働きはSOLVERメニューに似ていて、メニュー・キーを押すと、スタックから値を取り出して対応する変数に保存します。詳細は、第27章の「対話型プログラム」を見てください。
- リスト内に名称とコマンドが並んだもの（最初の要素がSTO以外のもの）が入っていると、MENUは普通のユーザ定義メニューを作ります。このメニューはUSERメニューとコマンド・メニューのあいのこのような働きをします。例は、275ページの「ディレクトリ移動」を見てください。

第22章

コマンドのカタログ機能

第１章で、いくつかのコマンドの正しいつづりを調べるためと、＋関数の引き数の各種組み合わせを見るために、コマンドのカタログ機能を使いました。この章では、カタログ内で使える操作と、引き数の正しい組み合わせを見るためのUSAGEメニューを復習します。

[CATALOG]を押すと、ABC順で最初のコマンドのABORTの表示と、CATALOGメニューになります。

キー	説　明
NEXT	次のコマンドのカタログに進みます。このキーを押し続けると、キーを放すまで順々にカタログが進みます。
PREV	前のコマンドのカタログに戻ります。このキーを押し続けると、キーを放すまで順々にカタログが戻ります。
USE	USAGEメニューの表示（次ページ参照）に切り替わり、そのコマンドが使うスタックの引き数が見えるようになります。
FETCH	カタログから抜け出して、コマンド行にそのコマンド名を書き込みます。
QUIT	カタログから抜け出しますが、コマンド行は変化しません。

[ON]を押すと、カタログから抜け出して、コマンド行もクリアしてしまいます。

コマンドの探索

左側のキーボードのキーを押すと、その文字で始まるコマンドに切り替えることができます。

- 左側のキーボードの英字キーを押すと、その文字で始まる最初のコマンドにカタログが切り替わります。その文字で始まるコマンドがないと、その前の文字で始まる最後のコマンドのカタログに切り替わります。
- 英字以外のキー（例えば 右シフト [Σ]）を押すと、その文字で始まる最初のコマンドにカタログが切り替わります。その文字で始まるコマンドがないと、英字以外で始まる最初のコマンドの＋にカタログが切り替わります。
- 右シフト [MENUS] を押すと、カタログの最後にある→STR にカタログが切り替わります。

コマンド使用法の確認

カタログ表示中のコマンドに使うスタックの引き数の正しい型を調べることができます。 USE を押すとカタログの2番目の段階（使用法メニューと呼びます）に切り替わり、そのコマンド用の引き数の組み合わせの全部を見ることができます。そのコマンド用の引き数の組み合わせが2組以上あると、次のようなメニュー・キーが現れます。（コマンドの引き数の組み合わせが1種類だけなら、 NEXT と PREV は現れません。）

キー	説　明
NEXT	次の引き数の組み合わせの表示になります。
PREV	前の引き数の組み合わせの表示になります。
QUIT	普通のカタログ表示に戻って、そのコマンドを表示します。これで別のコマンドの表示に切り替えたり、もう一度 QUIT を押してカタログ機能から抜け出すことができます。

[ON] を押すと、使用法メニューと普通のカタログ表示の両方から抜け出して、その時点のコマンド行をクリアしてしまいます。

第23章

オブジェクトの評価

簡単なキーボード操作だけの計算から、複雑なプログラムまで、計算機のどんな操作でも、評価という動作が伴っています。例えば次のようなことがあります。

- コマンド行にオブジェクトを1個または2個以上キー入力してから [ENTER] を押すと、コマンド行はプログラムに翻訳され、それからこれが評価されます。
- 普通入力状態でUSERメニューのキーを押すと、対応する名称が評価されます。
- 1段階ずつの微分をしているときに、 [EVAL] を押すとスタックの1段目の演算式を評価します。
- SOLVERを使って数値解を求めるときには、変数EQ内に保存している手続きを繰り返して評価します。

計算機の操作を理解する一番簡単な方法は評価の延期と評価の強制という用語を使うことです。評価の延期という用語そのものは初めてですが、その働きは知っているはずです。引用符で囲んだ名称または数式を入れるためにオブジェクトの識別記号を入力すると、それはオブジェクトの評価を延期したい(つまり、そのオブジェクトをスタックに入れたい)と指示したことになります。

評価の延期はどんな計算装置でもプログラムの基本となる概念で、これが使えないとプログラムは書き込んだとたんに実行されてしまいます。HP-28Sではこれを文字式による計算にまで拡張したので、文字式計算に名称と数式をデータとして使うことができます。例えば、演算式を評価するときには、次のどちらかを選べます。すなわち式中の変数などについて微分したり、文字式のままで解くことができます。もちろん、その数値を計算することもできます。

この章では各種のオブジェクトを評価するとどうなるかを説明します。一般論として、次のオブジェクトのクラス（役割の分類）について考えることにします。

- データ・クラスのオブジェクト　実数、複素数、ビット整数、文字列、配列、リストはこの分類に入ります。データ・オブジェクトの値とはその内容そのものです。
- 名称クラスのオブジェクト　この分類はさらにグローバル名とローカル名に分けられます。名称の値とは一般的には変数の内容です。
- 手続きクラスのオブジェクト　この分類はさらに数式とプログラムに分けられます。手続きの値とは定義されている手続きを処理した結果です。

おおざっぱに言うと、この分類はオブジェクトを評価したときにどうなるかを定義したもので、評価の結果としてそれ自体、または変数の内容、あるいは手続きの結果が返ります。けれども、実際にはもっと複雑なので、オブジェクトの各分類ごとに以下に詳しく説明します。

データ・クラスのオブジェクト

これがオブジェクトで一番単純なクラスです。どのデータ・クラスのオブジェクトを評価しても、同一のオブジェクトを返すだけです。

リストは多目的のデータ用オブジェクトです。というのはその中にどんな型のオブジェクトでも入れることができるからです。名称のリストの場合では、リストにすることで名称の評価を防止しています。名称をリストから取り出して初めて評価することができます。

名称クラスのオブジェクト

一般的には、名称の値とは変数の内容です。ローカル名の評価の方が簡単なので最初にこれを説明し、次にグローバル名の評価を取り上げます。

ローカル名の評価

第19章で説明したように、ローカル変数を使うのはスタック操作を簡単にするためです。ローカル変数の目的は、(1)変数の内容をスタックから取り除いて片付けておいて、(2)必要なときにはいつでも変数の内容を返せるようにするためです。従って、ローカル名を評価するとかならず対応するローカル変数の内容がスタックに返ります。

グローバル名の評価

一般的には、グローバル名を評価すると対応するグローバル変数の内容を評価します。言い換えると、グローバル名の評価はそれが表すオブジェクトを評価したのと同じ効果があります。

この一般的規則には例外が二つあります。

- 指定した名称の変数がないと、その名称がスタックに返ります。定義してない名称だけの変数を「形式上の変数」と呼びます。
- 指定した名称の変数の内容が数式なら、その数式を評価しません。HP-28Sはこのようなオブジェクトを評価しないので、記号が入ったままの計算を続けることができるのです。評価をしたいときには、その数式をスタックの1段目に入れてEVALコマンドを実行します。(数値の結果になるまで数式を自動的に繰り返して評価させるには、→NUMを実行します。)

変数にデータ・クラスのオブジェクトが入っていると、変数名の評価はその変数の内容を単に呼び出すことと同じです。しかし、変数名を評価すると評価の連続の原因になることがあります。例えば、ある変数に名称が入っていて、その名称が2番目の変数名であって、その2番目の変数にも名称が入っていて、その名称が3番目の変数名であると、1番目の変数の名称を評価することは最終的には3番目の変数の内容を評価することになります。

注　記　Xという名の変数に名称'X'が入っているように、値がそれ自体の名称になるような変数を作らないでください。このような変数を評価すると無限ループの原因になります。(無限ループとは同じことを永久的に繰り返すために、計算機が応答しなくなることです。)無限ループを止めるためには、システム停止(ON ▲)を実行する必要があり、こうするとスタックもクリアされてしまいます。

同様に、別の変数を経由して循環的に定義する変数も作らないでください。このような変数を評価すると無限ループの原因になります。

手続きクラスのオブジェクト

普通は、手続きの「値」はそれを定義した処理の結果です。プログラムが最も一般的な手続きクラスのオブジェクトなので、まずこれを説明し、次に数式を取り上げます。

プログラムの評価

プログラムとはオブジェクトとコマンドが一連に並んでいるものです。この説明書では「プログラムの評価」と「プログラムの実行」という用語を同じ意味として使っています。一般的には、プログラムの評価とはそのプログラムの内容を順番に取り出して、各オブジェクトをスタックに入れ各コマンドで実行することです。次の2点も覚えておいてください。

- 引用符で囲んだ名称はスタックに入りますが、引用符で囲んでない名称は評価されます。57ページで説明したように、引用符で囲んだ名称は評価を延期することを表します。
- プログラム構造はそれ自体の規則に従って実行されます。第1部でいくつかのユーザ定義関数を書き込みましたが、これには「ローカル変数構造」が入っています。プログラム構造については第26章で説明します。

名称の評価とプログラムの評価の規則から、HP-28Sのプログラム作成上の基本的な考え方の一つが導き出されます。ここでの説明では、「プログラム」とは変数に保存したプログラム、「プログラム名」はプログラムが入っている変数名の意味です。

ここでの基本的な考え方は「構造化プログラム」と呼ばれます。これは一つの複雑な作業をいくつかの部分作業に分け、部分作業ごとに一つのプログラムを作ります。これで作業全体の流れを反映した、比較的に単純な主プログラムにすることができます。この主プログラム内に、部分作業ごとの引用符で囲まないプログラム名を入れておくことで、それぞれの部分作業を実行することができます。ある部分作業を2回以上実行したいときには、引用符で囲まないプログラム名をその回数だけ入れておきます。別の主プログラムでも同じ部分作業を使うなら、同じ方法でその部分作業を実行することができます。

構造化プログラムの例は、253ページの「数式を完全に展開して同類項をまとめる」と、257ページの「ビット整数の表示」、270ページの「統計データの中央値」にあります。

数式の評価

個々の数式は、引用符で囲んでない名称と関数だけが入っているプログラムと同等です。数式を評価すると、対応するプログラムを評価した（つまり、引用符で囲んでない名称を評価し、関数を実行した）のと同じ結果になります。これについてはReference Manualの"Evaluation of Algebraic Objects"にも説明があります。

名称の評価は、200ページの「グローバル名の評価」で説明したように、その名称の変数があるかどうかによって決まります。いくつかの例を取り上げます。

- 名称がユーザ定義関数を指していると、そのユーザ定義関数を組み込み関数と同じように使うことができます。数式を評価するとユーザ定義関数を実行することになります。ユーザ定義関数の引き数は、括弧で囲んでユーザ定義関数名の後に続けますが、これも数式の一部です。

- 名称がプログラムを指していて、そのプログラムがスタックから引き数を取り出さなくて、しかも常に1個の結果を返すときには、結果を間接的に指すためにプログラム名を使うことができます。数式の評価はプログラムの実行を引きおこし、実質的にはプログラム名は結果に置き換わることになります。この例は、262ページの「基本統計量」にあります。
- 名称がさらに2番目の数式を指していると、1番目の数式の評価は2番目の数式の評価の原因にはなりません。その代わりに、1番目の数式内のその名称が結果として2番目の数式に置き換わります。

関数の中で特別なのは関数＝で、これで演算式と等式を区別します。＝を実行すると解答モード（記号解または数値解）の違いによって1個の等式または数値が返ります。

- 記号解答のモードでは、等式の評価で新しい等式ができます。新しい左側の演算式は元の左側の演算式を評価した結果です。新しい右側の演算式は元の右側の演算式を評価した結果です。
- 数値解答のモードでは、等式の評価で元の左辺の演算式の数値的結果と元の右辺の演算式の数値的結果との差になります。

次の部分で解答モードを詳しく説明します。

関数の評価

関数を評価するときには、その動作はその時点の解答モード（記号または数値のどちらか）によって決まります。このモードは次の章の「各種のモード設定」でも取り上げます。

記号解答モード　これが指定省略時の状態で、このときには関数は記号の引き数を受け入れて、記号の結果を返します。指定省略時の状態とは、HP-28Sを工場から出荷した、またはリセットした直後の状態のことです。記号解答モードでの関数の働きは、名称や演算式を計算してもっと大きな演算式を作ってみるとはっきりわかります。

数値解答モード　これはプロットのときやSOLVERに使われているもう一つのモードです。この目的は関数から確実に数値で結果が出るようにするためです。このモードでは、関数は記号の引き数を繰り返して評価し、数値だけの引き数を受け入れて、数値の結果を返します。

→NUMを実行することで、オブジェクトから数値の結果が返るまで強制的に評価することができます。第5章でπから数値を返すためにこれを使いました。

注　記　数値解答モードでは、演算式'$X+Y$'が入っている変数Xのように、変数自体の名称が入っている変数を評価しないでください。このような変数を評価すると無限ループの原因になります。この無限ループを止めるには、システム停止（ON ▲）を実行する必要があり、これでスタックもクリアされます。

同様に、変数を循環的に定義している変数を作らないでください。循環的定義が入っている変数を評価しても、無限ループの原因になります。

第24章

各種のモード

モードを切り替えることで、いろいろな場合の操作の結果に影響を与えることができます。一部のモード、例えば角度単位（度またはラジアン単位）はメニュー・キーを押すことで切り替えることができます。そのモードに切り替えると、そのメニュー記号に白い正方形が現れます。例えば、ラジアン単位に切り替えると、 RAD のようにメニュー記号が変わります。

警告音のモード(出る、または出ない)のような、大部分のモードは、SF(フラグのセット）とCF（フラグのクリア）コマンドを使うことで、ユーザ・フラグのセットまたはクリアで切り替えることができます。例えば、フラグ51で警告音のモードを制御するので、51 SF. を実行することで警告音が出ないようにすることができます。(このフラグとは、計算機が動作する際に参照する目印のようなものです。)

この章では、各種モードがHP-28Sの操作にどう影響するかを説明し、対応するメニュー記号とフラグも記載します。それにそのモードに切り替えると現れる案内表示も記載します。それぞれのモードについて、先に記載した方が指定省略時のモードで、メモリ・ロスト（メモリ全消去）の直後はこちらになります。

一般的なモード

このモードは計算と警告音に影響があります。

角度単位モード

このモードで角度を表す実数が度単位であるかまたはラジアン単位であるかが決まります。これは三角関数の引き数と、逆三角関数の結果に影響します。

度単位モード（ DEG 、フラグ60をクリア） 角度を表す実数は度単位です。

ラジアン単位モード（ RAD 、フラグ60をセット、(2π)） 角度を表す実数はラジアン単位です。

警告音のモード

このモードにより、エラー発生時またはBEEPの実行時にHP-28Sから音が出るかどうかを制御します。

音が出る（フラグ51をクリア） HP-28Sから音が出ます。

音が出ない（フラグ51をセット） HP-28Sから音は出ません。

主値のモード

ISOLまたはQUADが返す答えは、あり得る全部の答えを代表するための任意の符号（＋1または－1）と整数（0、1、2、…）が必要なのが普通です。このモードで、ISOLまたはQUADが返す答えに任意の符号と整数を入れるかどうかが決まります。

主値にしない（フラグ34をクリア） ISOLとQUADが返す答えには、任意の符号用の変数s1、s2、…と、任意の整数用の変数n1、n2、…が入ります。

主値だけ（フラグ34をセット） ISOLとQUADが返す答えは、任意の符号として＋1、任意の整数として0を使ったものになります。

定数のモード

このモードは、記号定数（e, i, MINR, MAXR, or π）を評価するとその数値を返すかどうかに影響します。数値解答モード（フラグ36をクリア）では、定数のモードに関係なく、記号定数の評価ではその数値が返ります。

記号定数（フラグ35をセット） 記号定数を評価すると、記号の形のままのものが返ります。

数値定数(フラグ35をクリア)　記号定数を評価すると、その数値が返ります。

解答モード

その時点の解答モードによって、関数の引き数に記号を使って評価した結果が影響を受けます。

記号解答(フラグ36をセット)　関数に記号の引き数を与えると、関数は記号の結果を返します。

数値解答(フラグ36をクリア)　関数はいつでも数値の結果を返します。そこで、引き数が記号の関数を評価すると、数値の結果になるまで繰り返して評価します。定数の状態に関係なく、記号定数を評価してもその数値が返ります。

入力と表示のモード

このモードでオブジェクトの入力と表示が影響を受けます。

入力モード

その時点の入力モードによって、コマンド・キー、関数キー、ユーザ・メニュー・キーを押したときの結果に影響があります。[']または[SHIFT]["]、[«]を押したときに、入力モードは自動的に切り替わります。また[α]を押すことで手動で切り替えることができます。カーソルの色でその時点の入力モードがわかります。詳細は、第18章の「コマンド行」を見てください。

普通入力モード(枠だけのカーソル)　コマンド・キー、または関数キー、ユーザ・メニュー・キーを押したときに、コマンド行を実行します。

数式入力モード(横線が2本入っているカーソル)　コマンド・キーを押したときに、コマンド行を実行します。

文字入力モード(黒いカーソル)　[ENTER]を押したときにだけ、コマンド行を実行します。

重ね書きまたは挿入モード

カーソル機能メニュー内の INS を押すと、重ね書きモードと挿入モードとの間で切り替わります。カーソルの外形で重ね書きモードまたは挿入モードのどちらになっているかがわかります。

重ね書きモード（四角形のカーソル）　新しい文字で既存の文字を書き換えます。

挿入モード（矢印形のカーソル）　新しい文字が既存の文字の間に挿入されます。

大文字と小文字

LC を押すと、大文字と小文字モードの間で切り替わります。

大文字モード　文字キーを押すと、コマンド行に大文字を書き込みます。

小文字モード　文字キーを押すと、コマンド行に小文字を書き込みます。

スタックの1段目の表示

オブジェクトによっては1行では表示できないほど大きいことがあります。必要に応じて、1段目のオブジェクトを複数行で表示するか、オブジェクトの大きさに関係なく1行だけを使うかを、切り替えることができます。この切り替えはトレースモードでの印字出力に影響します。

複数行使用（ ML 、フラグ45をセット）　1段目のオブジェクトが必要に応じて2行以上で表示されます。

複数行不使用（ ML 、フラグ45をクリア）　1段目のオブジェクトが1行だけで表示されます。

小数点のモード

コンマまたは点は、小数の分離記号（数値の整数部分と小数部分を区別するための記号、基点記号）と数値の区切り記号（コマンド行のオブジェクトを区切る記号。空白はいつでも区切り記号です）の役割をします。この両方の役割りをコンマと点に割り当てることができます。

小数の分離記号はコンマでない（ RDX. 、フラグ48をクリア）　点が小数点分離記号（小数点）で、コンマが区切り記号です。

小数の分離記号はコンマである（ RDX, 、フラグ48をセット）　コンマが小数点分離記号で、点が区切り記号です。

数値形式

このモードで実数の小数部分を何桁表示するかが決まります。FIX、SCI、ENGコマンドには実数の引き数*n*が必要です。（*n*は 0 ≦*n*≦11の整数で、整数でないときには整数に四捨五入してから使用されます。）その時点の数値形式はRND（round、四捨五入）コマンドに影響があります。

STD形式（ STD ）　実数は必要なときにだけ小数点付き、または指数部付きで表示されます。

FIX形式（ FIX ）　実数は*n*桁の小数部分付きで表示されます。必要なときにだけ指数部も表示されます。

SCI形式（ SCI ）　実数は、*n*桁の小数部分付きの10より小さい仮数と、指数部付きで表示されます。

ENG形式（ ENG ）　実数は、*n*＋1桁の仮数と、3の整数倍の指数部付きで表示されます。

ビット整数の基数

ビット整数の入力と表示用の基数（または底）を切り替えることができます。基数を切り替えても、ビット整数の内部表現（いつもビットが並んでいるものとして扱っています）には影響がありません。

十進法（ DEC **）**　基数記号なしで入力したビット整数は、十進数として解釈されます。十進法のときには、全部のビット整数は十進数で表示され、右端にdの基数記号が付きます。

16進法（ HEX **）**　基数記号なしで入力したビット整数は、16進数として解釈されます。16進法のときには、全部のビット整数は16進数で表示され、右端にhの基数記号が付きます。

八進法（ OCT **）**　基数記号なしで入力したビット整数は、八進数として解釈されます。八進法のときには、全部のビット整数は八進数で表示され、右端にoの基数記号が付きます。

二進法（ BIN **）**　基数記号なしで入力したビット整数は、二進数として解釈されます。二進法のときには、全部のビット整数は二進数で表示され、右端にbの基数記号が付きます。

ビット整数のワード長

ビット整数のワード長は1ビットから64ビットの範囲内で変えることができます。これでビット整数をどう表示するかが決まります。同様に、ビット整数を引き数として使う、または結果として返すときにはワード長より上位の部分を切り捨ててしまいます。ワード長を*n*に設定するには、*n* STWS（store wordsize、ワード長の保存）を実行します。

回復機能のモード

回復機能のモードによって、コマンド行、スタック、コマンドの引き数をコピーして保存するかどうかが決まります。間違えたときには、このコピーで間違える前の状態に回復できます。

CMDのモード

このモードで、ENTER（またはENTERを実行するキー）を押したときに、コマンド行のコピーを保存するかどうかが決まります。

CMDが働く（ CMD **）**　COMMANDで回復できるように、コマンド行のコピーを保存します。

CMDが働かない（ CMD ）　コマンド行を保存しません。

UNDOのモード

このモードで、 ENTER （またはENTERを実行するキー）を押したときに、スタックのコピーを保存するかどうかが決まります。

UNDOが働く（ UNDO ）　 UNDO で回復できるように、スタックのコピーを保存します。

UNDOが働かない（ UNDO ）　スタックを保存しません。

LASTのモード

このモードで、コマンドを実行したときに、その引き数のコピーを保存するかどうかが決まります。

LASTが働く（ LAST 、フラグ31をセット）　LASTまたはエラーのときに回復できるように、引き数のコピーを保存します。

LASTが働かない（ LAST 、フラグ31をクリア）　引き数は保存しません。エラーが発生しても、最後のコマンドの引き数はスタックに返りません。

数学的例外

0で割ろうとしたときのように、普通の実数計算中に発生するエラーの一部を、数学的例外と呼びます。普通のエラーのように計算が止まるようにもでき、または指定省略時の値を使って計算を続けることもできます。

結果が無限大のときの働き

結果が無限大のときの数学的例外は無限大の結果が返る計算のときに発生します。例えば、'LN(0)'、'TAN(90)'（角度が度単位のとき）、'X/0'.の評価などです。

結果が無限大のときにエラーになる（フラグ59をセット）　結果が無限大のときの数学的例外はエラーになります。

結果が無限大のときにエラーにならない（フラグ59をクリア）　結果が無限大のときの数学的例外は±9.99999999999E499を返して、無限大結果標識（フラグ64）をセットします。

桁あふれのときの働き

絶対値がこの機械で表現できる最大値よりも大きな有限値になるような結果になる計算をしようとすると、桁あふれの数学的例外が発生します。例えば、'9E499+9E499'、'EXP(5000)'、'FACT(2000)'.の評価などです。

桁あふれでエラーにならない（フラグ58をクリア）　この数学的例外は±9.99999999999E499を返して、桁あふれ標識（フラグ63）をセットします。

桁あふれでエラーになる（フラグ59をセット）　この数学的例外はエラーになります。

下位桁あふれの働き

絶対値がこの機械で表現できる最小値よりも小さな有限値になるような結果になる計算をしようとすると、下位桁あふれの数学的例外が発生します。例えば、'1E-499/2' または 'EXP(-5000)'.の評価などです。

下位桁あふれでエラーにならない（フラグ57をクリア）　この数学的例外は指定省略時の値0を返します。実際の結果の符号によって、＋側下位桁あふれ標識（フラグ62）または－側下位桁あふれ標識（フラグ61）をセットします。

下位桁あふれでエラーになる（フラグ57をセット）　この数学的例外はエラーになります。実際の結果の符号によって、Negative Underflow またはPositive Underflow のエラー・メッセージが出ます。

印字のモード

次のモードを使いわけることで、印字がかなり自由になります。

トレース印字

150ページのように計算記録を自動的に印字することができます。新しく作ったプログラムの試験などに便利です。

トレース印字をしない（ TRAC 、フラグ32をクリア）　自動的な印字はしません。

トレース印字をする（ TRAC 、フラグ32をセット）　コマンド行を実行するごとに、コマンド行の内容と、実行した操作やコマンド、それにスタック１段目の結果を印字するようになります。

自動改行のモード

普通は１回のコマンドでプリンタにデータを送っては印字します。一方、図形の印字などのように、印字しないでプリンタ内にデータを蓄積したいことがあります。このモードで、印字用コマンドが自動的に印字するかどうかが決まります。普通では印字用コマンドの種類によって、この状態を自動的に切り替えているので、これを考慮する必要はありません。

自動改行（フラグ33をクリア）　印字用コマンドはデータ伝送の終わりごとに改行命令を送るので、送られたデータをその場で印字するようになります。

印字データを蓄積（フラグ33をセット）　印字用コマンドは改行命令なしでデータを送るので、プリンタのバッファにデータを蓄積するようになります。（ただしバッファにデータがちょうどいっぱいになると、自動的に１行分印字します。）

印字速度

計算機はプリンタのバッファがあふれないように次のデータを送ってもよいかどうかを調べることができないので、データを安全に送る速さを計算します。このモードで、プリンタの電源が電池なのか、ACアダプタなのかに応じた動作をするように決まります。

普通の印字速度（フラグ52をクリア）　計算機は、電池を電源にした印字速度に適した速さで、データを送ります。

速い印字速度（フラグ52をセット）　計算機は、ACアダプタを電源にした印字速度に適した速さで、データを送ります。

印字の行間隔

このモードで空白行を自動的に作るかどうかが決まります。

1 行ごとの印字（フラグ47をクリア）　自動的な空白行は作りません。

2 行ごとの印字（フラグ47をセット）　文字行と文字行の間に空白行を自動的に作ります（1 行おきに印字します。）

第25章

本体の制御操作

この章では、普通のHP-28S動作に割り込んで実行する特別なキー操作を説明します。この本体制御操作は、表示画面の印刷(ハード・コピー)、表示の濃淡の調節、プログラムの強制停止、メモリのリセット、診断試験の実行などです。

どの本体制御操作も [ON] キーを押すことから始まります。[ON] を押したままで [DEL] を押すことで、どの本体制御操作でも取り消すことができます。

下表は本体制御操作用のキー操作を表したもので、その後にそれぞれの説明があります。

本体制御操作

目的	キー操作
表示画面の印字	[ON] [L]
表示の濃淡の調節	[ON] [+] または [ON] [−]
中断	[ON]
システム停止	[ON] [▲]
メモリのリセット	[ON] [INS] [▶]
反復試験(自己診断)	[ON] [◀]
キーボード試験	[ON] [NEXT]
本体制御操作の取り消し	[ON] [DEL]

表示画面の印字

表示画面を印字するには次のように操作します。

1. [ON] を押し続けます。
2. [L] (上側にPRINTと印刷してあるキー) を押します。
3. [ON] を放します。

表示の濃淡の調節

表示の濃淡を変えるには次のように操作します。

1. [ON] を押し続けます。
2. [+] を押すと濃くなり、[−] を押すと淡くなります。[ON] を押し続けている間は、[+] または [−] を繰り返えして押すか押し続けて、最適の濃さにすることができます。
3. [ON] を放します。

クリア操作

クリア操作には 3 段階あって、後のものほど影響範囲が広くなります。

中断

普通のスタック表示に戻るには、[ON] を押して中断します。場合によっては [ON] を 2 回押すことが必要になります。

中断の効果は次の通りです。

- コマンド行をクリアしてしまいます。
- どんなコマンドまたは手続きの実行でも取り消してしまいます。
- FORM、PLOT、カタログのような特別の操作から抜け出します。
- 普通のキーボード操作が再開始できます。

システム停止

[ON]では応答しないプログラムを強制的に止めるには、次のようにしてシステム停止を実行します。

1. [ON]を押し続けます。
2. [▲]を押します。
3. [ON]を放します。

システム停止の効果は次の通りです。

- 中断操作のときと同じ効果があります。
- 一時停止しているプログラムとローカル変数がクリアされます。
- スタック全体がクリアされます。
- 回復(CMD、UNDO、LAST)用に保存した内容がクリアされます。
- ユーザ定義メニューがクリアされます。
- その時点のディレクトリがHOMEに切り替わります。
- カーソル機能メニューが働くようになります。
- トレース印字モードではなくなります。

メモリのリセット

メモリ全体をリセットするには次のように操作します。

1. [ON]を押し続けます。
2. [INS]と[▶]を押し続けます。
3. [INS]と[▶]を放します。
4. [ON]も放します。

メモリをリセットすると次の効果があります。

- 中断とシステム停止の操作と同じ効果があります。
- 作成した変数とディレクトリの全部が消えます。
- ユーザ・フラグの全部が初期値に戻ります。
- 警告音が出て、1行目に `Memory Lost` の表示が出ます。

機能試験操作

計算機機能の試験用の本体制御操作が 2 種類あります。1 番目は電子回路の反復試験で、途中の操作は不要です。2 番目はキーボードの試験で、指定されている順番に全部のキーを押すことが必要です。どちらの試験もシステム停止を実行します。

反復試験(自己診断)

反復試験(自己診断)を実行するには次のように操作します。

1. 試験の開始
   a. [ON] を押し続けます。
   b. [◀] を押す。
   c. [ON] を放します。
2. 縦線や横線、空白の画面、各種の模様などを表示した後に試験結果をしばらく表示して、試験を続行します。
   - メッセージの OK-28S は計算機が試験に合格したことを示します。
   - -1 FAIL のようなメッセージは計算機が試験に不合格なことを示します。数値は不合格の種類などを示します。
     試験を中断するためにキーを押すと、不合格メッセージが出ますが、これはキーを押す操作が加わったためです。この場合の不合格メッセージは計算機に問題があったという意味ではありません。
3. 試験から抜け出すには、次のようにシステム停止を実行します。
   a. [ON] を押し続けます。
   b. [▲] を押します。
   c. [ON] も放します。

キーボードの試験

キーボードの試験を実行するには次のように操作します。

1. 試験の開始
   a. [ON]を押し続けます。
   b. [NEXT]を押します。
   c. [ON]も放します。
2. 表示が KEYBOARD TEST. になります。
   a. [A] [B] [C] [D] [E] [F]と押して、左側のキーボードの1行目を試験します。
   b. 左側のキーボードの2行目から6行目までを同じ方法で試験します。
   c. [INS] [DEL] [▲] [▼] [◀] [▶]と押して、右側のキーボードの1行目を試験します。
   d. 同じ方法で右側のキーボードの2行目から6行目までを試験します。(正しい順序で[ON]も押します。こうしても試験を中断しません。)
3. 正しい順序でキーを押して、キーボードが正しく働くと、OK-28S の表示になります。 1 FAIL のようなメッセージは、キーボードを正しい順序で押さなかったか、計算機が不合格なことを示します。数値は不合格の種類などを示します。
4. [ON]を押します。

第3部

プログラミング

第26章

プログラム構造

大部分のプログラムは、手で実行する一連のキーボード計算と同等です。オブジェクトがスタックに入り、コマンドが実行され、目的の結果が出てきます。このようなプログラムは、普通にキーボードで実行するのと同じ順番に、オブジェクトとコマンドをそのまま書き並べたものです。しかし、プログラム内には単純なキー操作を超える機能も使うことができます。

例えば、第1部でローカル変数を作るプログラムを書き込みました。特別なコマンド→には、その後に1個以上の名称が続き、その後に手続きが続いていて、これをローカル変数構造と呼びます。コマンド→はキーボードから実行することはできなくて、その名称と手続きが入っているプログラム全体の中だけで見られます。

この章では、まずローカル変数構造を復習します。次に条件に合うかどうかを試し、その結果によってプログラムの実行が変わる、別のプログラム構造を説明します。このプログラム構造用のコマンドはどれもPROGRAM BRANCHメニュー内にあります。223ページの「条件文構造」の一番目の例では、この章の残り部分で説明する基本的考え方を紹介しているので、必ず読んでください。

ローカル変数構造

第1部でいくつかのユーザ定義関数を書き込みましたが、これがローカル変数構造の一番重要な用途です。ユーザ定義関数には2種の必要条件があって、次の通りです。

- 引き数を明確に示します。
- 正確に1個の結果を返します。

例えば、第5章で書き込んだユーザ定義関数COTは次の通りでした。

```
« → x 'INV(TAN(x))' »
```

このローカル変数構造は1個の引き数をローカル変数 x に保存し(1番目の必要条件を満たしています)、演算式'INV(TAN(x))'で評価しています(2番目の必要条件を満たしています)。第14章のユーザ定義関数O→Gには、演算式ではなくてプログラムが入っていますが、このプログラムは正確に1個の結果を返すので、O→Gも2番目の必要条件を満たしています。

この必要条件はユーザ定義関数にだけ適用します。一般的には、ローカル変数はスタック操作の代用として使っています。次の例は2個の数値の和と差を返します。2個の結果を返すので、これはユーザ定義関数とは言えません。

```
« → x y « x y + x y - » »
```

これ以外の例は、第28章のプログラムを見てください。そこでも、スタック操作を避けるためのローカル変数構造を、ユーザ定義関数よりも多く使っています。

条件文構造

条件文構造を使うことで、プログラムを指定した条件で試験し、試験した結果によって何らかの選択をすることができます。ここではまず条件文構造の例を取り上げます。これを使って、プログラム構造を一般的に説明し、その後で別の条件文構造も説明します。

変数xを使っているプログラムを書いていて、$(\sin x)/x$を計算したいものと仮定します。$x=0$のときに商が不定になるのが難問です。次の例は、$x \neq 0$のときには$(\sin x)/x$が返り、$x=0$のときには1が返ります。

```
IF X 0 ≠ THEN X SIN X / ELSE 1 END
```

このプログラムを実行したときの、この構文の働きは次のようになります。

1. IFコマンドは単に構文の開始点の目印です。THENコマンドの前のどこにでも置くことができます。
2. Xを評価して、結果がスタックに入ります。
3. 数値0がスタックに入ります。
4. ≠　コマンドは引き数として、Xの値と数値0を使います。
   - この引き数が「等しくない」なら、≠　は1を返します。
   - この引き数が「等しくない」でなかったら、≠　は0を返します。
5. THENコマンドは、その引き数として1または0を取ります。
   - 引き数が1なら、THENはELSEの直前までのプログラム(この例では `X SIN X /`)を評価します。
   - 引き数が0なら、THENはELSEからENDの直前までのプログラム(この例では1)を評価します。
6. プログラムの実行はEND コマンドの後に続きます。

個々の条件文構造を説明する前に、プログラム構造についての一般的なことをまず説明しておきます。

プログラム構造コマンド　IF、THEN、ELSE、ENDコマンドはどれもプログラム構造コマンドです。このコマンドの並ぶ順序と意味は英語の文章での使用法とほぼ同じです。プログラム構造コマンドは、普通のコマンドのように柔軟に使うことはできなくて、この章で説明する組み合わせのときにだけ働きます。

条件テスト用関数とコマンド　関数≠を条件テスト用関数と呼びます。2個の数値を与えると、≠はテストをして真(その通り)か偽(そうではない)を表す1または0を返します。これ以外の条件テスト用関数は、<, ≤, ≥,> そして == です。(= は等式用に使っているので、テスト用には使えません。)記号の引き数を与えると、条件テスト用関数は記号の結果を返します。

このほかにいつも1または0を返す条件テスト用コマンドがあります。例えば、コマンドSAMEは == と似ていますが、こちらは単に2個のオブジェクトが同じかどうかを試験します。これ以外の試験用コマンドはフラグと一緒に使うもので、次で説明します。条件テスト用関数とコマンドの詳細は、Reference Manualの "PROGRAM TEST" を見てください。

フラグ　条件用関数とコマンドが返した数値の1と0をスタック・フラグと呼びます。これは試験の結果が真または偽だったかを表すので、1を真フラグ、0を偽フラグと呼ぶこともあります。(フラグとは旗の意味で、一種の標識として使っています。)

「フラグ」という用語は組み込みのユーザ・フラグをも指します。こちらには1から64までの番号が付いています。フラグ1から30まではユーザ自身の定義で真と偽の区別を表すのに使いますが、フラグ31から64までは計算機にとって特定の意味があります。スタック・フラグをユーザ・フラグに保存することができます。例えば、次の手順はA<Bならフラグ12をセットし、A≧Bならフラグ12をクリアします。

```
IF A B < THEN 12 SF ELSE 12 CF END
```

次の手順でフラグ12をセットしてあるかどうかをテストすることができ、これで元のテストのA B <と同じ真偽値を返します。

```
IF 12 FS? THEN ...
```

この技法の長所は、AとBの値が変化しても、元のテスト結果が保存されていることです。ユーザ・フラグの変更とテスト用のコマンドは、Reference Manualの〝PROGRAM TEST〟を見てください。この章の残り部分では、フラグとはスタック・フラグを指すことにします。

文節　2個のプログラム構造コマンドの間にあるオブジェクトとコマンドを文節と呼びます。プログラム構造は各文節を1個のまとまりのように扱います。文節はその論理上の役割またはその前に位置しているコマンドによって識別されます。223ページの最下行の例で説明すると次のようになります。

- IFとTHENの間の文節(`X 0 ≠`)をテスト文節またはIF文節と呼びます。
- THENとELSEの間の文節(`X SIN X /`)を真文節またはTHEN文節と呼びます。
- ELSEとENDの間の文節(1)を偽の文節またはELSE文節と呼びます。

223ページの例中の文節は単なる数値計算ですが、オブジェクトやコマンドのどんな組み合わせでも入れることができます。実質的には、文節はプログラム内の部分的プログラムのようなものです。文節を別々のプログラムに分けて、それを変数に保存すると、その文節として変数名を使うことができます。こうすると、複雑な条件から2個の複雑な結果に分かれるような構造でも、次のように変数A、B、Cの内容で決まるような、見やすい構造になります。

```
IF A THEN B ELSE C END
```

IF ... THEN ... ELSE ... END

上で説明したばかりの用語を使うと、この条件文構造の評価は次のように表すことができます。IF文節を評価すると、フラグが返ります。そのフラグが真なら、THEN文節を評価します。そのフラグが偽なら、ELSE文節を評価します。

この構造の別の例は、248ページの〝FIB 2〟(フィボナッチ数、ループ版)を見てください。

IFTE (If-Then-Else-End関数)

223ページの例を関数IFTEを使って数式の形で書くと次のようになります。

```
'IFTE(X≠0,SIN(X)/X,1)'
```

この形の方が記号処理には簡単です。xを定義してない(変数Xにまだ代入してない)ときに、このプログラム構造を実行すると、この数式のままのものが結果になります。IFTEの引き数は数式の形で表すことができることが必要です。条件テスト部分にはRPN コマンドを入れても良く、このプログラム構造の形になるように使います。

IFTH関数は、247ページの〝FIB1〟(フィボナッチ数、再帰版)に使っています。

IF ... THEN ... END

ELSE文節が不要なら(つまり、「するか、しないか」のどちらかを選択したいとき)、プログラム構造からELSE文節を取り除くことができます。次の例は1段目のオブジェクトが2段目のオブジェクトよりも大きいときにだけ、両方のオブジェクトを入れ替えます。

```
IF DUP2 ≤ THEN SWAP END
```

DUP2を使うのは2個のオブジェクトをコピーするためです。コピーした方は次の比較≤で使って消えてしまいます。IF ... THEN ... ENDの別の例は、270ページの"SORT"(リストの並べ換え)を見てください。

IFT(IF-THEN-ENDコマンド)

前例のプログラム構造の代わりにIFT コマンドを使っても書けます。

```
DUP2 ≤ « SWAP » IFT
```

DUP2と≤でスタックに1個のフラグが入り、プログラム« SWAP »もスタックに入ります。IFTコマンドは引き数としてフラグとプログラムを使います。フラグが真なら、IFTはプログラムを評価し、フラグが偽なら、IFTは引き数を捨てます。結果はプログラム構造形式と同じです。

エラーの捕促(エラー・トラップ)

場合によってはプログラム実行中にエラー発生が予想できることがあります。普通はエラーが発生するとプログラムの実行が打ち切りになります。しかしプログラム構造の中にエラーを捕まえるコマンドを入れておくと、エラーが発生してもプログラムの実行を続けることができます。

(sin x)／xのプログラムでは、$x=0$のときに無限大エラーが発生しました。(sin 0)／0＝1を定義する別の方法は次の通りです。

```
IFERR X SIN X / THEN DROP2 1 END
```

これはIFERR文節(% SIN % /)を評価して、エラーが発生したら、THEN文節(DROP2 1)を評価する、という意味です。

この例にはDROP2コマンドが入っているのでエラー原因の2個の0を捨てます。これはLASTが使えることが前提です。LASTが使えないときには、0がないのでDROP 2 コマンドは不適当です。エラートラップをするときにはLASTの状態を確認してください。

IFERR ... THEN ... ENDの別の例は259ページの〝BDISP(ビット整数の表示)〟にあります。次の形式を使うと、エラーが発生しないときにだけ評価するELSE文節を入れることができます。

```
IFERR ... THEN ... ELSE ... END
```

指定回数ループ構造

ループ構造には繰り返して評価するループ文節を入れます。指定回数ループ構造では、ループ文節を何回評価するかをプログラムで指定します。条件停止ループ構造と呼ぶ別のプログラム構造では、ループ文節の評価を繰り返すかどうかを決めるテスト文節を使います。ここでは指定回数ループ構造を説明して、条件停止ループ構造は231～232ページで説明します。

START ... NEXT

次の例は音が4回出ます。

```
1 4 START 440 .1 BEEP NEXT
```

この構造は次のように働きます。

1. STARTコマンドはスタックから数値の1と4を取り出してカウンタ(回数を記憶しておくもの)を作ります。このカウンタはループを何回繰り返したかを確認するのに使います。数値1はカウンタの開始値、数値4は最終値の指定です。

2．ループ文節の440　.1　BEEPを実行します。
3．NEXTコマンドはカウンタに1を加えます。
4．この時点のカウンタ値を最終値と比較します。
   - この時点のカウンタ値が最終値を超えていなかったら、手順の2～4を繰り返します。
   - この時点のカウンタ値が最終値を超えていたら、指定回数ループ構造を抜け出して、NEXTコマンドの後にプログラム実行が続きます。

この例では、手順2～4を4回繰り返します。ループ・カウンタは最初は1から2に増え、次に3、次に4、そして5に増えます。この時点でカウンタが最終値4を超えるので、指定回数ループが終わります。判定前に手順1を実行するので、どんなときにでもループ文節を少くとも1回は実行します。START ... NEXTの別の例は248ページの"FIB2(フィボナッチ数, ループ版)"にあります。

FOR カウンタ名 ... NEXT

大部分の場合ではその時点のカウンタ値をループ文節の変数として使えると便利です。そのために、START をFOR名称に置き換えます。カウンタは指定した名称のローカル変数になります。以前に、本書ではグローバル変数と区別するためにローカル変数名は小文字で書くとよいと説明しました。次の例は1から始まる5個の整数の二乗値がスタックに入ります。

```
1 5 FOR × × SQ NEXT
```

FOR ×の部分は1回だけ実行します。× SQ はループ文節で、これを繰り返し実行します。

この例ではカウンタの開始値を1に指定しましたが、どんな整数でも受け付けます。カウンタを変数として使うので、希望する開始値と最終値になるようにカウンタ値の開始値と最終値を決めます。次の例は3から9までの各整数の二乗値がスタックに入ります。

```
3 9 FOR × × SQ NEXT
```

別の例は、259ページの〝BDISP(ビット整数の表示)〟を見てください。

... 増分 STEP

NEXTコマンドはカウンタを1ずつ増やすときに使います。別の増減値を指定するには、NEXTをn STEPに置き換えます(nは希望する増減値です)。次の例のように、STEPをFOR カウンタの後に使うのが普通ですが、START の後にも使えます。次の例は1^2から5^2までの奇数の整数の二乗値がスタックに入ります。

```
1 5 FOR × × SQ 2 STEP
```

ループ文節の × SQ 2 は3回実行します。STEP コマンドはカウンタを最初は1から3に、次に5に、次に7に増やします。この時点でカウンタ値が最終値の5を超えるので、指定回数ループ構造から抜け出します。

上の例ではカウンタ値が増加しました。カウンタ値を減らすには、負数の増分を指定します。次例は5^2から1^2までの奇数の整数の二乗値がスタックに入ります。

```
5 1 FOR × × SQ -2 STEP
```

-2 STEP の部分でカウンタが5から3に、次に1、次に－1に減ります。この時点でカウンタ値が終了値の1よりも小さくなるので、指定回数ループ構造から抜け出します。

270～272ページの〝SORT(リストのソート)〟プログラムでは、カウンタを1ずつ減らすために -1 STEP を使っています。この場合は、NEXTと同じように、STEPがカウンタ値を1ずつ変えていますが、カウンタは増えるのでなく減っています。

条件停止ループ構造

ループの繰り返し回数を前もって指定できないときには、ループ文節とテスト文節の両方が入っている条件停止ループ構造を書きます。両文節を交互に実行し、テスト文節の結果によって続けるかどうかが決まります。

ここでは2種類の条件停止ループ構造を説明します。最初のは、DO ... UNTIL ... ENDで、テスト文節の前にループ文節を実行します。そこで、いつでもループ文節を最低1回は実行します。2番目は、WHILE ... REPEAT ... ENDで、テスト文節を最初に実行します。そこで、場合によってはループ文節を実行しないことがあります。

DO ... UNTIL ... END

次の例はオブジェクトが変化しなくなるまで、オブジェクトを繰り返し評価します。最初の試験をする前に1回評価したいので、この例ではDO ... UNTIL ... ENDを使います。

```
DO DUP EVAL UNTIL DUP ROT SAME END
```

この構造は次のように働きます。

1. ループ文節の DUP EVAL を実行すると、オブジェクトとそれを評価した結果がスタックに残ります。
2. テスト文節の DUP ROT SAME を評価すると、評価結果とフラグがスタックに残ります。フラグはオブジェクトとそれを評価した結果が同じかどうかを表します。
3. スタックからフラグを取り出します。この値でループ文節を繰り返すかどうかが決まります。
   - フラグが偽なら、手順(1)～(3)を繰り返します。
   - フラグが真なら、ループ構造を抜け出します。

ここでは 'A+B' を完全に評価したいものと仮定し、Aには 'P+Q' がPには2が入っているものとします。ループ文節の最初の評価で 'A+B' と 'P+Q+B' がスタックに返ります。両式は同じでないので、ループ文節を2回目に評価すると 'P+Q+B' と '2+Q+B' が返ります。この式も同じでないので、ループ文節を3回目に評価すると、'2+Q+B' と '2+Q+B' が返ります。両式が同じなので、ループ構造を抜け出します。

→NUMでは名称が未定義の場合にエラーになることを除くと、この例の効果と→NUMの効果は同じです。この例のもっと応用範囲が広いものは、253ページの「連続実行」を見てください。

WHILE ... REPEAT ... END

次の例はスタックから何個かのベクトルを取り出して、それをその時点の統計用行列に加えます。加える前に1段目のオブジェクトがベクトルかどうかを試験したいので、この例はWHILE ... REPEAT ... ENDを使っています。

```
WHILE DUP TYPE 3 == REPEAT Σ+ END
```

この構造は次のように働きます。

1. テスト文節の DUP TYPE 3 == を評価すると、スタックにフラグが1個入ります。このフラグは2段目のオブジェクトが実数のベクトルかどうかを表します。
2. このフラグをスタックから取り出します。この値によってループ文節を実行するかどうかが決まります。
   - フラグが真なら、ループ文節のΣ+を実行して、統計用行列にベクトルを加えて、手順(1)と(2)を繰り返します。
   - フラグが偽なら、ループ構造を抜け出します。

WHILE ... REPEAT ... ENDはフラグが偽のときに終わりますが、DO ... UNTIL ... ENDはフラグが真のときに終わることに注意してください。テスト文節の真偽値を変えたいときには、最後のコマンドとしてNOTを加えて、WHILE ... NOT REPEATまたはUNTIL ... NOT ENDにします。

WHILE ... REPEAT ... END の別の例は257ページの「左側に空白を入れる」を見てください。

入れ子のプログラム構造

プログラム内の１文節は働きが副プログラムに似ているので、文節自体にプログラム構造を入れることができます。文節のさらに内側の構造を内方構造と呼び、文節が入っている構造を外郭構造と呼びます。270ページの「SORTリストの並べ換え」プログラムに入れ子の指定回数ループがあります。

プログラムが理解可能でありさえすれば、入れ子の回数(重なり回数)は無制限です。場合によっては、プログラムの内方構造を変数に記憶させ、外郭構造の文節でその変数名を使う方が簡単です。

第27章

対話型プログラム

実行中にユーザの指示が必要なプログラムもあります。変数にユーザからの入力が必要なときには、プログラムからユーザへ入力を頼むことができます。いくつかの選択肢の中からユーザの選択が必要なときには、プログラムからユーザへ選択を頼むことができます。

この章ではPROGRAM CONTROL メニューにある次のコマンドを使って、プログラムから入力または選択を頼む例を説明します。

コマンド	説明
HALT	プログラムの進行を止める。
s WAIT	プログラムの進行を*s*秒間止める。
KEY	1個のキーを押したら、そのキーの文字列を返します。
f *s* BEEP	周波数が*f*の音を*s*秒間鳴らします。
CLLCD	表示画面をクリアします。
n DISP	表示画面の*n*行目に1個のオブジェクトを表示します。
CLMF	プログラム実行が完了したときに普通の表示画面に戻します。

入力の依頼

次のプログラム手順で変数A、B、Cの入力用のユーザ定義メニューを作って、警告音でユーザに知らせ、入力できるように実行を停止します。

```
... « STO A B C » MENU 440 .1 BEEP HALT ...
```

表示されるメニューの記号は [A]、[B]、[C]で、これはSOLVERメニューに似ています。スタックに値を入れてから、このキーのどれかを押すだけで対応する変数にその値が入ります。値を入力した後に、プログラムの実行を再開するには ▒ [CONT] を押します。

選択の依頼

複雑な仕事には小さなプログラムをいくつか書いて、それぞれで小さな仕事を実行するのが最善です。場合によっては一部の仕事を実行するのにいくつかの選択肢を持っていることがあります。一つの方法として仕事を実行する別のプログラムをいくつか書くという手段があります。

ある仕事が完了したら、次の仕事用にHOP 、SKIP、JUMPプログラムの中からどれかを選ぶものと仮定します。次のプログラム手順はHOP、SKIP、JUMPプログラム用のユーザ定義メニューを作り、警告音を鳴らし、プログラムの実行が終了します。

```
...« HOP SKIP JUMP » MENU 440 .1 BEEP »
```

表示されるメニー記号は [HOP] 、[SKIP] 、[JUMP] で、普通のメニューに似ています。このメニュー・キーのどれかを押すと、次の仕事が始まります。選択肢の中から選んだプログラムで次の仕事が終わり、このようにして複雑な仕事全体が進みます。

もっと複雑な例

次の列はメッセージを表示し、ユーザがキーを押すまで待ち、そのキーが定義してある(つまり、選択が正しい)かを調べます。キーが定義してあれば、対応した動作を実行し、キーが未定義だと、エラー・メッセージを表示して最初に戻ります。

この例は前章で説明したプログラム構造を使っています。定義してあるキーを押すまで繰り返すために、外郭のDO ... UNTIL ... END構造があります。外側のDO文節には、キーを押すまで繰り返すために、内側にDO ... UNTIL ... END構造が入っています。外側のUNTIL文節には、未定義キーを押したらエラー・メッセージを出すためのテストが入っています。次のリストは外郭構造、その文節、内方構造、その文節とをそれぞれ区別しやすいように、行の開始位置をずらしてあります。

プログラム手順	注釈
`{ "Apple" "Banana" "Cherry" }`	このリストは出力用に準備します。そして次のDO ... UNTIL ... END 構造でユーザの選択を表す 1 ～ 3 が返るまでスタックに残ります。
`DO`	外側のループ文節の始まり。この文節で、ユーザの選択とその結果を説明する選択用のメッセージを表示します。
`CLLCD`	表示をクリアします。
`"Press"`	1行目の選択用メッセージ。
`1 DISP`	メッセージを 1 行目に表示。
`" [A] for Apple"`	2行目の選択用メッセージ。
`2 DISP`	メッセージを 2 行目に表示。
`" [B] for Banana"`	3行目の選択用メッセージ。
`3 DISP`	メッセージを 3 行目に表示。
`" [C] for Cherry"`	4行目の選択用メッセージ。
`4 DISP`	メッセージを 4 行目に表示。
`DO UNTIL KEY END`	内側の条件停止ループはユーザがキーを押すまで繰り返します。KEYコマンドはキーを押さないと0を返し、キーを押すとそのキーを表す文字列と 1 を返します。ループを抜け出したときには文字列がスタックに残ります。
`UNTIL`	外側のテスト文節の始まり。この文節で、押したキーが定義してあるキーかどうか調べます。

```
{ "A" "B" "C" }
```
定義したキーが入っているリストです。これは定義したキーと出力との間で1対1に対応しています。

```
SWAP POS
```
キー文字列と定義キーのリストを照合します。POSはキー文字列が "A" なら1、キー文字列が "B" なら2、キー文字列が "C" なら3、該当するものがないと0を返します。

```
 IF DUP
```
フラグとして使うために位置をコピーします。位置が 1、2、3 ならTHEN文節を実行します。位置が 0 なら、ELSE文節を実行します。

```
 THEN 1
```
キーが定義キーなら、スタックに真フラグを入れます。

```
 ELSE
```
キーが未定義キーなら、エラー・メッセージを表示して音を出します。

```
  CLLCD "Bad key"
  1 DISP
```
エラー・メッセージの表示

```
  440 .1 BEEP
```
音出し。

```
  1 WAIT
```
1秒間待ちます。

```
 END
```
IF ... THEN ... ELSE ... END構造の終わり。キーが定義キーなら、位置と真フラグがスタックにあります。キーが未定義キーなら、位置(これが偽フラグを兼ねています)だけがスタックにあります。

```
END
```
外側の条件停止ループの終わり。キーが定義キーなら、スタックに位置を置いてループを抜け出します。キーが未定義キーなら、ループ文節を繰り返します。

```
GET
```
出力結果のリストと位置をスタックから取り出して、対応する結果を返します。

EVAL	結果を評価します。この例では、結果が文字列なのでEVALは効果がありません。もっと実際的な例では、結果が(多分、変数内の)プログラムのことでしょうが、それの実行にはEVALが必要です。
CLMF	普通のスタック表示に戻します。

このプログラム手順を実行すると、次の選択メッセージが出ます。

[A]、[B]、[C]以外のキーを押すと、音が出てエラー・メッセージが1秒間だけ現れます。

そして選択メッセージになります。[A]、[B]、[C]のどれかを押すと、それに応じて1段目に文字列の"Apple"、"Banana"、"Cherry"が返ります。

出力用のリスト、選択メッセージ、定義キーのリストを変更すると、このプログラム手順がもっと意味のある文字列をスタックに入れることができます。もっと一般的にするには、ローカル変数を使って、上のプログラム手順をローカル変数構造の内側に入れます。次のプログラムになります。

```
« → keys m1 m2 m3 m4
 « DO CLLCD
    m1 1 DISP
    m2 2 DISP
    m3 3 DISP
    m4 4 DISP
    DO UNTIL KEY END
   UNTIL keys SWAP POS
    IF DUP
    THEN 1
    ELSE CLLCD "Bad key" 1 DISP
     440 .1 BEEP 1 WAIT
    END
   END
 »
 GET EVAL CLMF
»
```

このプログラムをKEY？という名の変数に入れて、次を実行すると前例と同じになります。

```
{ "Apple" "Banana" "Cherry" }
{ "A" "B" "C" }
"Press"
" [A] for Apple"
" [B] for Banana"
" [C] for Cherry"
KEY?
```

第28章

プログラム例

この章にはHP-28S用のプログラムが20あります。どれも便利ですが、重要なことは、各種プログラム技法の例になっていることです。どのプログラムにも次の記述があります。

- スタック説明図　スタック説明図は左右に分けてあって、引き数と結果を表します。引き数はプログラムを実行する前にスタックに入れておくことが必要なもので、結果はプログラムがスタックに残すものです。
  スタック説明図はスタックの一部だけを表しています。ユーザ・メモリを変更したり、オブジェクトを表示するプログラムではスタックに無関係のことがあります。
- 技法　これが一番重要な部分です。この章で使った技法を理解すると、自作プログラムに応用できます。
- 必要なプログラム　一部のプログラムは別のプログラムをサブルーチンとして呼び出します。呼び出されるプログラムと呼び出すプログラムを入れる手順は無関係ですが、呼び出すプログラムを実行する前に全部を入れておきます。
- プログラムと注釈　この章のプログラム・リストはプログラム構造と働きが分かるようにしました。プログラムを入力するときにはこの形式に合わせる必要はありません。しかし、リスト内の空白部分や別の行のオブジェクトとの間には空白をキー入力してください。

  プログラムは１文字ずつキー入力することも、コマンドごとにメニュー・キーを押して入力することもできます。結果がリストに合えばどちらでも違いはありません。

プログラムをキー入力するときには、プログラムの最終部分にある閉じ括弧と最後の識別記号は省略できます。 ENTER を押すと自動的に追加されます。

- 例　ここの例はSTD表示形式を前提にしています。STD表示形式に切り換えるには、STD ENTER と押すかMODEメニューを使います。

この章で取り上げた一番重要なプログラム技法は構造化プログラムで、小さなプログラムを使って別のプログラムを組み上げます。次のプログラムを別のプログラムの中で使っています。

- BOXSはBOXRの中で使っています。
- MULTIはEXCOの中で使っています。
- PADとPRESERVEはBDISPの中で使っています。
- ΣGETはΣX2、ΣY2、ΣXYの中で使っています。
- SORTとLMEDはMEDIANの中で使っています。

BOX関数

ここには次の2種のプログラムを載せました。

- BOXSは箱の全表面積を計算します。
- BOXRは箱の体積と表面積の比率を計算します。

BOXS(箱の表面積)

四角い箱の高さ、幅、長さを与えて、その箱の全表面積を計算します。

引き数	結果
3：高さ 2：幅 1：長さ	3： 2： 1：面積

技法

- ローカル変数構造　ローカル変数を使うとグローバル変数と衝突しないで、引き数に同じ名称を割り当てることができます。グローバル変数と同様に、スタック内の位置を気にしないで何回でも引き数が使えるので、ローカル変数は便利です。グローバル変数と違って、そのローカル変数を作ったプログラム構造が終わるとローカル変数は消えてしまいます。
ローカル変数構造は 3 個の部分に分かれます。

  1. →という名のコマンド。文字入力モードでこのコマンドをキー入力するときには、自動的に前後に空白が入ります。(他のコマンドと同様に、→もキーを押して入力しますが、前後が空白のときにだけこのコマンドになります。この 1 字コマンドを # や « のような識別記号と混同しないでください。)
  2. 1 個以上の名称。
  3. 名称が入っている手続き(数式、等式、またはプログラム)。この手続きを定義式とも呼びます。

  ローカル変数構造を評価すると、それぞれの名称のローカル変数が作られます。ローカル変数の値はスタックから取り出されます。次に定義手順を評価すると、ローカル変数はこの値で置き換わります。
  ローカル変数の威力を見るには、次のプログラムと244ページにあるプログラムを比較してください。
- ユーザ定義関数　この種のプログラムはRPN法と数式方式のどちらの入力でも働きます。ユーザ定義関数とはローカル変数構造ひとつで成り、1 個の結果だけを返すプログラムです。

プログラム	注釈
«	プログラムの始まり。
→ h w l	高さ、幅、長さ用のローカル変数を作ります。約束として、小文字を使います。値はスタックから(RPN法のとき)またはユーザ定義関数の引き数から(数式方式のとき)取り出します。

プログラム	注釈
'2*(h*w+h*l+w*l)'	表面積の定義式です。ユーザ定義関数を評価すると、この数式を評価して面積をスタックに返します。
»	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' BOXS STO	プログラムをBOXSとして保存。

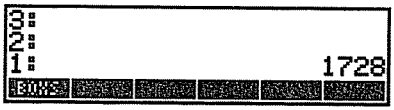
例　ユーザ定義関数の利点の一つはRPNと数式方式のどちらでも働くことです。高さ12cm、幅16cm、長さ24cmの箱の表面積を計算します。最初はRPNで、次は数式記法で計算してみます。

RPN法では、最初に高さと幅を入れます。

USER
12 ENTER
16 ENTER

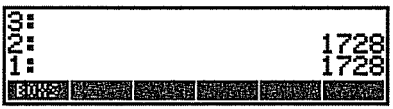
次に長さをキー入力してBOXSを実行します。

24 BOXS



表面積は1728cm²です。
今度は数式法でやってみましょう。

' BOXS (12,16,24 EVAL



やはり表面積は1728cm²です。

ローカル変数を使わないBOXS

次のプログラムはスタック操作だけを使って箱の表面積を計算します。このプログラムと前のBOXSを比較してください。

引き数	結果
3：高さ	3：
2：幅	2：
1：長さ	1：面積

プログラム	注釈
≪	プログラムの始まり。
DUP2 *	*wl*の計算。
ROT	*w*を1段目に移動。
4 PICK	*h*を1段目にコピー。
*	*wh*の計算。
+	*wl*＋*wh*の計算。
ROT ROT	*h*と*l*を2段目と1段目に移動。
*	*hl*の計算
+	*wl*＋*wh*＋*hl*の計算
2 *	2(*wl*＋*wh*＋*hl*)の計算
≫	プログラムの終わり。

このBOXSはユーザ定義関数ではないので、数式方式には使えません。

BOXR(箱の表面積と体積の比率)

箱の高さ、幅、長さを与えて、その表面積と体積の比率を計算します。

引き数	結果
3：高さ 2：幅 1：長さ	3： 2： 1：面積／体積

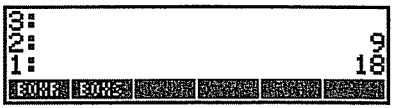
技法

- 入れ子になっているユーザ定義関数。BOXRはユーザ定義関数で、定義式の計算にはBOXSを使っています。さらに、BOXRは別のユーザ定義関数の中で定義するのにも使えます。
  呼び出すBOXSはローカル変数として*h*、*w*、*l*を使って定義していますが、BOXRの定義の中ではBOXSはその引き数として、*x*、*y*、*z*を使っています。それぞれの定義内のローカル変数はお互いに無関係なので、両方の定義内でローカル変数名が一致していても、一致していなくても違いは出ません。しかし、1個の定義内ではローカル変数名が首尾一貫することが重要です。

プログラム	注釈
≪	プログラムの始まり。
→ x y z	高さ、幅、長さ用のローカル変数を作ります。このプログラムでは*h*、*w*、*l*でなくて、*x*、*y*、*z*を使っています。
'BOXS(x,y,z)	ユーザ定義関数BOXSを使った定義の始まり。
/(x*y*z)'	箱の体積で割る。
≫	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' BOXR STO	プログラムをBOXRとして保管。

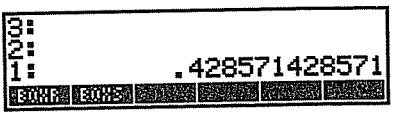
例　高さが9 cm、幅が18cm、長さが21cmの箱の表面積と体積の比率を計算してみましょう。最初はRPNで、次に数式方式で計算します。
RPNでは、最初に高さと幅を入れます。

USER
9 ENTER
18 ENTER



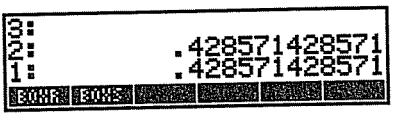
次に長さを入れてBOXRを実行します。

21 BOXR



比率は0.428571428571です。
今度は数式方式です。

' BOXR (9,18,21 EVAL



今度も、比率は0.428571428571です。

フィボナッチ数

整数nを与えて、n番目のフィボナッチ数Fnを計算します。フィボナッチ数は次のような数です。

$$F_0 = 1、F_1 = 1、F_n = F_{n-1} + F_{n-2}$$

ここでは2種類のプログラムを取り上げて、この問題の解決法を見ることにします。

- FIB1は再帰的(循環的)定義のユーザ定義関数です。再帰的定義とは定義式の中にそれ自体の名称が入っている定義です。FIB1は短くて、理解しやすいのが利点です。
- FIB2は指定回数ループのプログラムです。FIB1より長くて複雑ですが、計算が早いのが利点です。

FIB1(フィボナッチ数、再帰版)

引き数	結果
1 : *n*	1 : *Fn*

技法

- IFTE(If-Then-Else関数)。　FIB1の定義式には条件判断関数のIFTEが入っていて、これはRPNにも数式方式にも使えます。(FIB2はプログラム構造のIF ... THEN ... ELSE ... ENDを使っています。)
- 再帰　FIB1の定義式内にFIB1そのものを使っていて、ここでF_{n-1}とF_{n-2}を使ってFnを定義しています。

プログラム	注釈
≪	プログラムの始まり
→ n	ローカル変数の定義
'	定義式の開始
IFTE(n≤1,	もしもn≦1なら、
n,	そのときはFn=n、
FIB1(n-1)+FIB1(n-2))	そうでないならFn=F_{n-1}+F_{n-2}。
'	定義式の終わり。
≫	プログラムの終わり。
ENTER	スタックにプログラムを入れる。
' FIB1 STO	FIB1としてプログラムを保存

例　RPNでF_6を計算し、数式方式でF_{10}を計算しましょう。

まずRPNでF_6を計算します。

USER
6 FIB1

次に数式方式でF_{10}を計算します。

' FIB1 (10 EVAL

FIB2(フィボナッチ数、ループ版)

引き数	結果
1 : *n*	1 : Fn

技法

- IF ... THEN ... ELSE ... END。FIB2では条件判断にこのプログラム構造を使っています。(FIB1はIFTEを使いました。)
- START ... NEXT(指定回数ループ)。Fnを計算するのに、FIB2はF_0とF_1から出発して、次のF i を計算するループを繰り返します。

プログラム	注釈
≪	プログラムの始まり
→ n	ローカル変数を作る。
≪	定義プログラムの始まり。
IF n 1 ≤	もしもn≦1なら、
THEN n	そのときはFn=n、
ELSE	ELSE文節の始まり。
0 1	F_0とF_1をスタックに入れる。
2 n	2からnまで、
START	次のループを実行します。
DUP	最後のF(最初はF_1)をコピーする。
ROT	その前のF(最初はF_0)を1段目に移す。
+	次のF(最初はF_2)を計算します。
NEXT	ループを繰り返す。
SWAP DROP	F_{n-1}を捨てます。
END	ELSE文節の終わり。
≫	定義プログラムの終わり。
≫	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' FIB2 STO	ログラムをFIB2として保存。

例　F_6とF_{10}を計算しましょう。FIB2の方がFIB1よりもずっと速く計算できます。
F_6を計算しましょう。

USER
6 FIB2

F_{10}を計算しましょう。

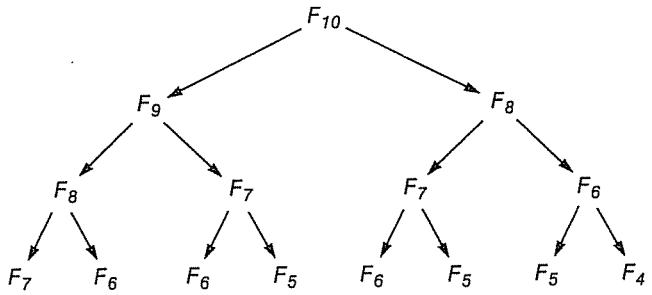
' FIB2 (10 EVAL

FIB1とFIB2との比較

FIB1は中間結果のF_iを2回以上計算していますが、FIB2は各中間結果F_iを1回だけ計算します。そのため、FIB2の方が速くなります。

FIB1の計算時間は*n*の指数関数的に長くなりますが、FIB2の計算時間はnに比例するだけなので、*n*が大きくなるほど計算時間の差が大きくなります。

下図はFIB1でF_{10}の計算の開始部分を示したものです。中間結果の計算回数に注意してください。1行目は1回、2行目は2回、3行目は4回、4行目は8回になります。



1ステップずつの実行

プログラム内容を1ステップずつ実行して、それぞれのスタックの影響を見ると、プログラムの働きの理解が簡単になります。こうすると自作のプログラムのデバッグ(間違いの発見)が楽になり、あるいは他人が書いたプログラムが理解しやすくなります。

ここではFIB2を1ステップずつ実行してみますが、どんなプログラムにもこの規則が使えます。一般的規則は次の通りです。

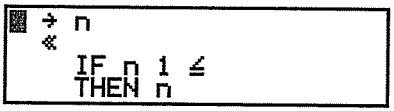
1. VISIT を使ってプログラム内にHALTコマンドを挿入します。1ステップずつの実行を始めたい場所にHALTを入れます。(FIB2に入れたHALTの位置による実行の影響は後で見られます。)
2. プログラムを実行します。HALTコマンドを実行すると、プログラムが停止します。(停止の案内表示で分かります)。
3. PROGRAM CONTROL メニューに切り替えます。
4. SST を1回押すと、次のプログラム・ステップをちょっとの間表示してから実行します。
   ここで次のことができます。
   - SST を押して次のステップを見て実行する。
   - CONT を押してそれ以後を普通に実行する。
   - KILL を押してそれ以後のプログラム実行を放棄する。
5. そのプログラムを普通に実行したくなったときには、VISITを使ってそのプログラムからHALTを取り除いてください。

最初の例として、FIB2内の最初のコマンドとしてHALTを挿入します。
スタックをクリアしてUSERメニューに切り換えます。

[CLEAR] [USER]

VISITを使ってFIB2をコマンド行に戻します。

['] [FIB2] [VISIT]



HALTコマンドを挿入します。

[▶] [INS] [CONTROL] [HALT]

FIB2の変更版を保存します。

[ENTER]

F_1を計算します。最初は、案内表示の●が現れる以外は何も変わりません。

[USER]
1 [FIB2]

PROGRAM CONTROLメニューに切り換えて、SST(single-step、1ステップ)を実行します。実際に実行する前に表示の1行目に何が現れるか注目してください。

[CONTRL] [SST]

→ n で1ステップです。この1ステップとは単純にプログラム内の次のオブジェクトを指しているのではなく、論理的な単位です。

前々ページの一般的規則をもう一度見てください。もう4まで進んでいるので、3つの方法の中から選んでください。
この例では、FIB2が途中であることを示す、案内表示の◙が消えるまで SST を繰り返して押してください。(この操作例は省略します。)
F_1の計算ではFIB2のTHEN文節だけを実行しました。2番目の例として、3 FIB2を実行してF_3の計算を1ステップずつ進めましょう。今度はSTART … NEXTループが入っているELSE文節を実行します。n＝3なので、START … NEXTループを2回実行するのが見られます。
3番目の例では、START … NEXTループをまとめて1回ずつ見ることにします。つまりFIB2内やループ自体内の全ステップを1ステップずつ実行するのではなく、ループの各繰り返しの前にスタックを見ます。そのために、HALTコマンドをループの内側に動かします。次にループに到着するまではFIB2が止まらないようにしたいので、 CONT (continue、続行)を使うとループを1回ずつ実行できます。
VISITを使ってFIB2をコマンド行に戻します。

USER
' FIB2 VISIT

```
HALT → n
  «
    IF n 1 ≤
    THEN n
```

カーソル機能を使ってHALTを削除します。次に図のように、STARTコマンドのすぐ後にHALTを挿入します。

```
    IF n 1 ≤
    THEN n
    ELSE 0 1 2 n
      START HALT◆DUP R…
```

変更後のFIB2を保存します。

ENTER

```
3:
2:
1:
FIB2 FIB1 BOX# BOX2
```

F_3の計算を始めましょう。FIB2はループを実行する前に止まります。

3 FIB2

```
3:
2:                              0
1:                              1
FIB2 FIB1 BOX# BOX2
```

ループの実行を続けましょう。FIB2は2回目のループを実行する前に止まります。

CONT

ループの実行を続けます。これがループの最後の実行なので、FIB2の実行が完了します。

CONT

FIB2の実験が終わったときに、VISITを使ってHALTコマンドを削除するのを忘れないでください。

数式を完全に展開して同類項をまとめる

ここには2個のプログラムを載せました。

- MULTIはプログラムが効果を発揮しなくなるまでプログラムを繰り返します。
- EXCOはMULTIを使って数式を完全に展開して同類項をまとめます。

MULTI(連続実行)

1個のオブジェクトとそのオブジェクトを扱うプログラムを与えると、オブジェクトが変化しなくなるまでそのプログラムを繰り返して適用します。

引き数	結果
2：オブジェクト 1：《 プログラム 》	2： 1：結果のオブジェクト

技法

- DO ... UNTIL ... END(条件停止ループ)。DO文節には繰り返したい部分を入れます。UNTIL文節には両方の文節を再び実行するか(偽のとき)または抜け出す(真のとき)かを決めるテスト部分を入れます。
- 引き数としてのプログラム。一般にプログラムには名称を付けて、後でその名称を呼び出して実行しますが、プログラムをスタックに入れて別のプログラムの引き数として使うこともできます。
- ローカル変数の評価。引き数のプログラムを繰り返して実行するにはローカル変数に保存します。事前に何回コピーする必要があるのか分からないときには、オブジェクトをローカル変数に保存すると便利です。

MULTIはグローバル変数とローカル変数の違いの1例です。グローバル変数に名称またはプログラムが入っているときに、変数名を評価すると変数の内容を評価します。しかしローカル変数の内容は単に呼び出すだけです。そこで、MULTIは変数名を使って引き数のプログラムをスタックに入れて、次に明示したEVALコマンドでプログラムを評価します。

プログラム	注釈
≪	プログラムの始まり。
→ p	引き数のプログラムを入れるローカル変数pを作ります。
≪	定義プログラムの始まり。
DO	DO文節の始まり。
DUP	オブジェクトをコピーする。
p EVAL	そのオブジェクトを引き数としてプログラムを実行させると、新しいオブジェクトを返します。ローカル変数はその内容を評価しないでスタックに返すだけなので、プログラムを実行するにはEVALコマンドが必要です。)
UNTIL	UNTIL文節の始まり
DUP	新しい方のオブジェクトをコピーする。
ROT	古い方のオブジェクトを1段目に移す。
SAME	古い方と新しい方が同じかどうかテストする。

プログラム	注釈
END	UNTIL 文節の終わり。
≫	定義プログラムの終わり。
≫	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' MULTI STO	プログラムをMULTIとして保存する。

例　MULTIは次のプログラム内で使っています。

EXCO(数式を完全に展開して同類項をまとめる)

数式オブジェクトを与えて、数式が変化しなくなるまで繰り返してEXPANを実行し、その後で数式が変化しなくなるまで繰り返してCOLCTを実行します。場合によっては結果が1個の数値になることもあります。

引き数	結果
1：数式	1：数式
1：数式	1：*z*

技法

- 構造化プログラミング手法　EXCOはプログラムMULTIを2回呼び出します。MULTIをどこにも使わないようにもできますが、MULTIを別のプログラムとして作っておいて、MULTIを2個所に入れる方が、MULTI内の全コマンドを繰り返して入れるよりもずっと能率的です。

必要なプログラム

- MULTI(253ページ)　EXCOが引き数として渡すプログラムを繰り返して実行します。

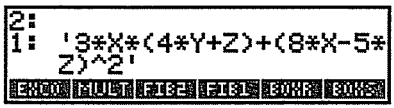
プログラム	注釈
≪	プログラムの始まり。
≪ EXPAN ≫	スタックにEXPANを入れる。（コマンドが1個のプログラムです。）
MULTI	数式オブジェクトが変化しなくなるまでEXPANを実行する。
≪ COLCT ≫	スタックにCOLCTを入れる。（コマンドが1個のプログラムです。）
MULTI	数式オブジェクトが変化しなくなるまでCOLCTを実行する。
≫	プログラムの終わり。
ENTER	スタックにプログラムを入れる。
' EXCO STO	EXCOとしてプログラムを保存。

例　次の数式を展開して同類項にまとめます。

$$3x(4y+2)+(8x-5z)^2$$

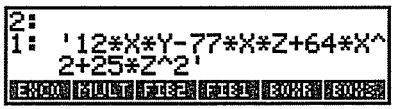
まず数式を入れます。

USER
' 3 × x × (4 × y + z ▒) + (8 × x − 5 × z ▒) ▒ ^ 2
ENTER



完全に展開して同類項をまとめます。

EXCO



多項式になるほどEXPANを何回も反復実行して数式を完全に展開するので、EXCOの実行時間は長くかかります。

ビット整数の表示

ここには次の3プログラムを載せました。

- PADはオブジェクトを右端に寄せて表示する実用的な補助プログラムです。
- PRESERVEは計算機のモード(角度単位、ビット整数の基数、など)を変更するプログラム内に使える便利な補助プログラムです。
- BDISPはビット整数をHEX、DEC、OCT、BINで表示するプログラムです。このプログラムは数値を右寄せして表示するためにPADを呼びだし、ビット整数の基数を保存するためにPRESERVEを呼び出します。

PAD(左側に空白を入れる)

オブジェクトを文字列に変換し、その文字列が22文字以下であったら、その前に空白を挿入します。

DISPを使って短い文字列を表示すると、左寄せで表示するので、最初の文字が左端になります。最後の文字位置は文字列の長さで決まります。

短い文字列の前に空白を挿入することで、PADは文字列全体を右側に動かします。空白を含む文字列が23文字になると、右寄せしたようになって、最後の文字が表示の右端になります。

PADは23文字より長い文字列には効果がありません。

引き数	結果
1：オブジェクト	1：〝オブジェクト〞

技法

- WHILE ... REPEAT ... END(条件停止ループ)。WHILE文節には、REPEAT文節を実行してもう一度テストする(真のとき)、またはREPEAT文節を飛び越えて抜け出す(偽のとき)かをテストする部分が入っています。

■ 文字列操作 PADにはオブジェクトを文字列に変換し、文字列の長さを数え、2個の文字列を結合する方法を使っています。

プログラム	注釈
≪	プログラムの始まり。
→STR	オブジェクトを確実に文字列の形にします。(文字列はこのコマンドの影響を受けません。)
WHILE	WHILE文節の始まり。
DUP SIZE 23 <	文字列が23文字よりも短いかを調べる。
REPEAT	REPEAT文節の始まり。
" " SWAP +	文字列の前に余白を付ける。
END	REPEAT文節の終わり。
≫	プログラムの終わり。
ENTER	スタックにプログラムを入れる。
' PAD STO	プログラムをPADとして保存

例 PADはBDISPプログラム内で使っています。

PRESERVE(以前のフラグ状態を保存して元に戻す)

別なプログラムをスタックに入れておいて、このプログラムを実行すると、実行前のフラグ設定状態を保存し、スタックのプログラムを実行し、その後で以前のフラグ設定状態に戻します。

引き数	結果
1： ≪ プログラム ≫	1：(プログラムの結果)

技法

■ RCLFとSTOF PRESERVEは計算機のその時点のフラグ状態をビット整数の形で記録するためにRCLF(recall flags、フラグの呼び出し)と、ビット整数の形から以前の状態に戻すためにSTOF(store FLAGS、フラグの保存)を使っています。

■ ローカル変数構造　PRESERVEはスタックから一時的にオブジェクトを取り除くためにローカル変数を１個作ります。この定義プログラムはスタック上の引き数のプログラムを評価することが主な役目です。

プログラム	注釈
≪	プログラムの始まり。
RCLF	64個のユーザ・フラグ全体の設定状態を表す64ビットのビット整数を呼び出す。
→ f	ビット整数をローカル変数fに保存。
≪	定義プログラムの始まり。
EVAL	引き数のプログラムを実行
f STOF	64個のユーザ・フラグの状態を戻す。
≫	定義プログラムの終わり。
≫	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' PRESERVE STO	プログラムをPRESERVEとして保存。

例　PRESERVEはプログラムBDISP 内で使っています。

BDISP(ビット整数の表示)

数値を16進、10進、８進、２進法で表示します。(小数部分があると、小数第１位を四捨五入して表示。)

引き数	結果
1 : #n	1 : #n
1 : n	1 : n

技法

■ IFERR ... THEN ... END(エラーの検知)　実数にも使えるように、BDISPにはR→B(real-to-binary、実数からビット整数に)コマンドを入れてあります。しかし、このコマンドは引き数がビット整数のときにはエラーが発生します。

エラーが発生したも実行が続けられるように、R→BコマンドはIFERR文節の中に置きました。エラーが発生したときには処理が不要なので、THEN文節にはコマンドがありません。

- LASTを使えるようにする　エラーが発生したときに、引き数をスタックに戻すためにはLASTが使えることが必要です。BDISPではフラグ31をセットして、プログラム的にLASTの回復機能が使えるようにしています。
- FOR ... NEXTループ(カンウタを使った指定回数ループ)　BDISPは1から4までループを実行して、各回ごとに別の行に引き数nを別の基数で表示します。
  ループ・カウンタ(このプログラムではjと名付けました)はローカル変数です。この変数を作ったのは→コマンドではなくて、FOR ... NEXTプログラム構造です。この値はNEXTで自動的に増加します。
- 副プログラム　BDISPでは副プログラムの使用法を3種類利用しています。
  1. BDISPには実行の基本にする副プログラムが1個とPRESERVEの呼び出し部分が入っています。基本にする副プログラムはスタックに入り、PRESERVEでこれを評価します。
  2. BDISPはローカル変数*n*を作るので、これに続く定義プログラムも副プログラムとなります。
  3. ループ内で特別の働きをする副プログラムが4個あります。各副プログラムにはビット整数の基数を変えるコマンドが1個入っていて、ループの繰り返しごとにこの副プログラムから1個だけ実行します。

必要なプログラム

- PAD(257ページ)は文字列を右端に表示するように23文字の長さにしています。
- PRESERVE(258ページ)はその時点のフラグ状態を保存して、基本の副プログラムを実行し、そして以前のフラグ状態に戻します。

プログラム	注釈
≪	プログラムの始まり。
≪	基本副プログラムの始まり。
DUP	nをコピーする。
31 SF	LASTが使えるようにフラグ31をセット。
IFERR	エラー・トラップの始め。
R→B	nをビット整数に変換。
THEN	エラーが発生したら、
END	何もしない(THEN文節にコマンドがないからです)。
→ n	ローカル変数nを作る。
≪	定義プログラムの始まり。
CLLCD	表示部をクリア。
≪ BIN ≫	BIN用の副プログラム。
≪ OCT ≫	OCT用の副プログラム。
≪ DEC ≫	EDC用の副プログラム。
≪ HEX ≫	HEX用の副プログラム。
1 4	カウンタの開始値と終了値。
FOR j	カウンタjを使ったループの始まり。
EVAL	基数の副プログラムを1個評価する(最初はHEX)。
n →STR	nをこの時点の基数で文字列にする。
PAD	文字列が23文字になるように余白を入れる。
j DISP	j行目に文字列を表示。
NEXT	jを1増やして、ループを繰り返す。
≫	定義プログラムの終わり。
≫	基本副プログラムの終わり。
PRESERVE	この時点のフラグ状態を保管して、基本副プログラムを実行し、以前のフラグ状態に戻す。
≫	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' BDISP STO	プログラムをBDISPとして保存。

例　基数を10進法に切り替えて、#　100を全基数で表示し、BDISPが基数を10進法に戻したかどうか調べてみましょう。
スタックをクリアして、BINARYメニューに切り替えます。

[CLEAR]
[BINARY]

基数をDECにして、#　100をキー入力します。

[DEC]
[#] 100 [ENTER]

BDISPを実行します。(次の操作でBINARYメニューを見たいので、メニューを切り替えないでください。)

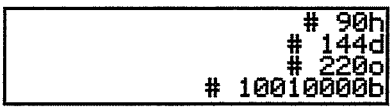
BDISP [ENTER]

普通のスタック表示に戻して、基数を調べてみましょう。

[ON]

基本の副プログラムでは計算機をBINにしたままですが、PRESERVEでDECに戻しています。
実数でもBDISPが働くかを調べるために、144で試してみましょう。

[USER]
144 [BDISP]



基本統計量

2変量の統計計算では、2個の変数の平方和(Σx^2とΣy^2)と積和(Σxy)を計算しておくと便利なことがあります。ここには次の5プログラムを載せました。

- SUMSは変数ΣCOVを作ります。その内容はその時点の統計用行列からの共分散行列です。
- ΣGETはΣCOV内の指定した位置の数値を取り出します。
- ΣX2はΣGETを使ってΣCOVからΣx^2を取り出します。
- ΣY2はΣGETを使ってΣCOVからΣy^2を取り出します。
- ΣXYはΣGETを使ってΣCOVからΣxyを取り出します。

ΣDATの列数がnなら、ΣCOV はn×n行列になります。プログラムΣX2、ΣY2、ΣXYはΣPAR(統計パラメータ)を調べてx値が入っている列(C_1と呼ぶ)とy値が入っている列(C_2と呼ぶ)を決めます。

技法

- 行列操作　このプログラムではどうやって行列を転置し、2個の行列を掛けて、行列から数値を1個取り出すかをお見せします。
- 数式オブジェクト内に使えるプログラム　ΣX2、ΣY2、ΣXYは数式記法に従っている(スタックからは引き数を取り出しませんが、結果を1個だけスタックに入れます)ので、この名称を普通の変数のように数式や等式内に使うことができます。
- ΣPARの規則　いくつかの2変量用統計コマンドはΣPARという名の変数を使って、ΣDAT内のどの2個の列を使うかを指定します。ΣPAR内には4個の数値が入ったリストが1個あって、最初の2個で列を指定します。(残りの2個は回帰直線の傾斜とy切片です。)
  SUMSは0 PREDV DROPを実行して必ずΣPARが存在するように準備しています。PREDV(predicted value、予想値)コマンドはΣPARがまだない場合は、指定省略時の値を使ってΣPARを作り出し、DROPは0のときの予想値を捨てます。
  ΣX2、ΣY2、ΣXYはΣPAR内に保存している値を使ってどの要素をΣCOVから取り出すかを決めています。

SUMS(統計量行列)

統計用行列ΣDATから共分散行列を作り、これを入れた変数ΣCOVを作ります。

例として、ΣDATが次のようなn×2行列とすると、

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

ΣCOVには次の共分散行列が入ります。

$$\begin{bmatrix} \Sigma x^2 & \Sigma xy \\ \Sigma xy & \Sigma y^2 \end{bmatrix}$$

引き数	結果
1：	1：

プログラム	注釈
≪	プログラムの始まり。
RCLΣ	n×mの統計行列ΣDATの内容を呼び出す。
DUP	コピーを作る。
TRN	行列を転置する。結果はm×nの行列。
SWAP *	両方の行列を掛けてm×mの共分散行列を作る。(行列を入れ替えないと、積がn×nの行列になります。)
'ΣCOV' STO	共分散行列を変数ΣCOVに保存。
0 PREDV DROP	ΣPARがないときにはΣPARを作り出す。
≫	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' SUMS STO	プログラムをSUMSとして保存。

ΣGET(ΣCOVの要素を1個取り出す)

ΣPARの1番目として2番目の位置を指すpとqを与えて、ΣCOVのrs番目の要素を取り出します。このrとsはΣPAR内の1番目と2番目の要素に対応したものです。

ΣX2、ΣY2、ΣXYから次の引き数と一緒にこのΣGET を呼び出します。

- ΣX2では、p＝1とq＝2。
- ΣY2では、p＝2とq＝2。
- ΣXYでは、p＝1とq＝2。

引き数	結果
2：1または2	2：
1：1または2	1：ΣCOVのrs番目の要素

プログラム	注釈
«	プログラムの始まり。
ΣCOV	スタックに共分散行列を入れる。
ΣPAR	スタックに統計用パラメータ行列を入れる。
DUP	コピーする。
5 ROLL	*p*を1段目に移す。
GET	ΣPAR内の*p*番目の要素に対応した*r*を取り出す。
SWAP	ΣPARを1段目に移す。
4 ROLL	*q*を1段目に移す。
GET	ΣPARの*q*番目の要素に対応した*s*を取り出す。
2 →LIST	{*r*, *s*} をスタックに入れる。
GET	ΣCOVから*rs*番目の要素を取り出す。
»	
ENTER	プログラムをスタックに入れる。
' ΣGET STO	プログラムをΣGET として保存。

ΣX2(xの平方和)

Σx^2を計算します。このxはC_1列(ΣPAR内の1番目の要素で指定した列)の要素です。

引き数	結果
1：	1：Σx^2

プログラム	注釈
≪	プログラムの始まり。
1 1	C_1を2回指定。
ΣGET	Σx^2を取り出す。
≫	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' ΣX2 STO	プログラムをΣX2として保存。

ΣY2(yの平方和)

Σy^2を計算します。このyはC_2列(ΣPAR内の2番目の要素では指定した列)の要素です。

引き数	結果
1：	1：Σy^2

プログラム	注釈
≪	プログラムの始まり。
2 2	C_2を2回指定。
ΣGET	Σy^2を取り出す。
≫	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' ΣY2 STO	プログラムをΣY2として保存。

ΣXY(xとyの積和)

Σ*xy*を計算します。この*x*と*y*はC_1とC_2列(ΣPAR内の1番目と2番目の要素で指定した列)の要素です。

引き数	結果
1：	1：Σ*xy*

プログラム	注釈
≪	プログラムの始まり。
1 2	C_1とC_2の指定。
ΣGET	Σ*xy*を取り出す。
≫	
ENTER	プログラムをスタックに入れる。
' ΣXY STO	プログラムをΣXYとして保存。

例　次の統計行列のΣX2、ΣY2、ΣXYを計算しましょう。

$$\begin{bmatrix} 18 & 12 \\ 4 & 7 \\ 3 & 2 \\ 11 & 1 \\ 31 & 48 \\ 21 & 17 \end{bmatrix}$$

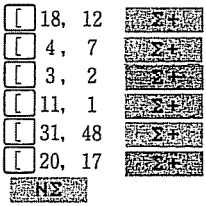
一般的な手順は次の通りです。

1. 統計データを入力する。
2. SUMSを実行して共分散行列ΣCOVを作ります。
3. ΣX2、ΣY2、ΣXYを実行します。
4. ΣDAT内に要素が3列以上あったら(つまり、各データが3変数以上なら)、
   a. COL Σを実行してC_1とC_2用の新しい値を指定する。この値がΣPAR内に入ります。
   b. ΣX2、ΣY2、ΣXYを実行します。

さて上例を試してみましょう。

スタックをクリア、STATメニューに切り替え、ΣDATをクリアします。

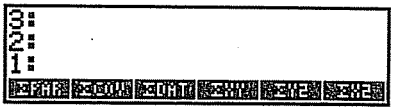
データを入力し、入力したデータが全部で6行かどうか調べてみましょう。



データ数は捨てます。

共分散行列ΣCOVを作ります。

USER
SUMS

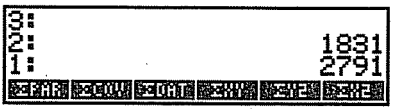


Σx^2を計算しましょう。

Σ X2

Σy^2を計算しましょう。

Σ Y2



Σxyを計算しましょう。

Σ XY

統計用行列が3列以上なら、C_1とC_2の新しい値が指定できます。この例では、現在値の$C_1 = 1$と$C_2 = 2$を指定しましょう。

COLΣコマンドはSTATメニュー内にありますが、コマンド名のキー入力が簡単なので、USERメニューのままにしておきましょう。

1 ENTER
2 SPACE COLΣ ENTER

これで新しいC_1とC_2列でのΣX2、ΣY2、ΣXYが実行できます。

統計用行列ΣDATに新しいデータを追加したり削除したときには、またSUMSを実行するのを忘れないでください。

統計データの中央値(メジアン)

ここには次の3プログラムがあります。

- SORTはリスト内の要素を昇順に並べ換えます。
- LMEDは並べ換えたリストの中央値を計算します。
- MEDIANはSORTとLMEDを使って、その時点の統計用行列の中央値を計算します。

SORT(リストの並べ換え)

リストを昇順に並べ換えます(整列します)。

引き数	結果
1：{リスト}	1：{並び換えたリスト}

技法

- バブル・ソート(並べ換えの方法の一種です。)　リスト内の1番目と2番目の数値から始めて、SORTは隣り合った数値を比較して、大きい方の数値をリストの後側に移します。この方法をリストの最後まで順次繰り返すと、リスト内の最大値がリストの最後に移ります。これをもう1回繰り返すと2番目に大きな数値がリストの最後から2番目に移ります。これをリスト内の数値の個数に応じた回数繰り返します。
- 入れ子の指定回数ループ(二重になっている指定回数ループ)　外側のループは各回の並べ換え停止位置を制御しています。内側のループは1番目から停止位置までの並べ換えを実行します。
- 入れ子のローカル変数構造　SORTには二つのローカル変数構造があって、2番目の内側に1番目の定義プログラムが入っています。この入れ子は便宜上のものです。2個の値を算出してまとめて2個のローカル変数を作るよりも、最初の値を算出してすぐにローカル変数を作り、それに関連した値をスタックから捨てる方が簡単です。

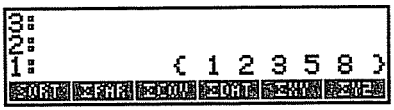
■ FOR … STEPとFOR … NEXT(指定回数ループ)　SORTでは2個のカウンタを使っています。外側のループでは-1 STEPで各繰り返しごとにカウンタを1ずつ減らし、内側のループではNEXTで各繰り返しごとにカウンタを1ずつ増やしています。

プログラム	注釈
≪	プログラムの始まり。
DUP SIZE 1 - 1	最後の一つ手前の位置から最初の位置まで、
FOR j	カウンタjで外側のループを始める。
1 j	最初の位置からj番目の位置まで、
FOR k	カウンタkで内側のループを始める。
k GETI → n1	リスト内のk番目の数値を取り出して、それをローカル変数n_1に保存する。
≪	外側の定義プログラムの始まり。
GETI → n2	リスト内の次の数値を取り出して、それをローカル変数n_2に保存する。
≪	内側の定義プログラムの始まり。
DROP	カウンタを捨てる。
IF n1 n2 >	数値2個の順番が不適当なら、
THEN	次のようにします。
k n2 PUTI	後側の数値をk番目の位置に入れる。
n1 PUT	k番目にあった数値を次の位置に入れる。
END	THEN文節の終わり。
≫	内側の定義プログラムの終わり。
≫	外側の定義プログラムの終わり。
NEXT	kを増やして内側のループを繰り返す。
-1 STEP	jを減らして外側のループを繰り返す。
≫	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' SORT STO	プログラムをSORTとして保存。

例

{8、3、1、2、5}のリストを並べ替えてみましょう。

USER
{ 8,3,1,2,5 SORT



LMED(リストの中央値)

並べ替えたリストを与えて、その中央値を計算します。リスト内の要素数が奇数のときには、中央値は中心の要素の値です。リスト内の要素数が偶数のときには、中央値は中心の直前と直後の要素の値の平均値です。

引き数	結果
1：{並べ替えたリスト}	1：並べ替えたリストの中央値

技法

- FLOORとCEIL　整数値では、FLOOR とCEILはどちらもその整数を返します。整数でない実数では、FLOORとCEILはその数値のすぐ外側の整数(小数部分を切り捨てた整数と小数部分を切り上げた整数)を返します。

プログラム	注釈
≪	プログラムの始まり。
DUP SIZE	リストの大きさ(要素数)。
1 + 2 /	リストの中心位置(端数は要素数が偶数のリスト用)。
→ p	中心位置をローカル変数pに保存します。
≪	定義プログラムの始まり。
DUP	リストをコピーする。
p FLOOR GET	中心位置または中心の直前の数値を取り出す。
SWAP	リストを1段目に移す。
p CEIL GET	中心位置または中心の直後の数値を取り出す。

プログラム	注釈
+ 2 /	中心位置の2数または中心の直前直後の2数の平均。
»	定義プログラムの終わり。
»	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' LMED STO	プログラムをLMEDとして保存。

例

SORTを使って並べ替えたリストの中央値を計算しましょう。

USER
LMED

LMEDはプログラムMEDIANが呼び出して使います。

MEDIAN(統計データの中央値)

統計データの各列の中央値を表すベクトルを返します。

引き数	結果
1：	1：[x_1、x_2 … x_m]

技法

- 配列、リスト、スタックの要素。MEDIANはΣDAT からデータをベクトルの形で1列分取り出します。ベクトルからリストに変換するために、MEDIANはベクトルの要素をスタックに入れて、次に組み合わせて1個のリストにしています。そしてSORTとLMEDを使ってこのリストから中央値を計算します。
  まずm番目の列の中央値を計算し、1列目の中央値を最後に計算するので、中央値を計算するごとに、以前に計算した中央値がスタックの上方に移ります。

全中央値を計算した後には、中央値がスタックに正しい順番に入っているので、これを1個のベクトルに組み合わせます。

- FOR ... NEXT(カウンタを使った指定回数ループ)　MEDIANはループを使って各列ごとに中央値を計算します。中央値を逆順に(最後の列を最初に)計算するので、中央値の順序とは逆にカウンタを使っています。

必要なプログラム

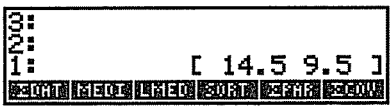
- SORT(270ページ)　リストを昇順に並べ換えます。
- LMED(272ページ)　順番に並べ換えたリストの中央値を計算します。

プログラム	注釈
≪	プログラムの始まり。
RCLΣ	その時点の統計用行列ΣDATをこれから加工するので、元のデータを保存するためにコピーをスタックに入れます。
DUP SIZE	リスト{*n*　*m*}をスタックに入れる。このnはΣDAT 内の行数で、mは列数です。
LIST→ DROP	nとmをスタックに入れる。リストの大きさ(要素数)は捨てる。
→ n m	nとm用のローカル変数を作る。
≪	定義プログラムの始まり。
'ΣDAT' TRN	ΣDATを転置する。これからnはΣDAT内の列数を、*m*は行数を表します。
1 m	最初と最後の行。
FOR j	各行ごとに、以下の処理をします。
Σ-	ΣDAT内の最後の行を取り出す。最初はこれが*m*行目で、これが元のΣDATのm列目に相当します。
ARRY→ DROP	行の要素をスタックに入れる。リストの指標{*n*}を捨てます。このnはローカル変数にもう入れてあるから不要です。
n →LIST	要素が*n*個のリストにする。
SORT	リストを昇順に並べ換えます。
LMED	リストの中央値を計算します。

プログラム	注釈
j ROLLD	中央値をスタックの正しい位置に移す。
NEXT	jを増してループを繰り返します。
m 1 →LIST	リスト {*m*} を作る
→ARRY	全部の中央値を組み合わせて、要素がm個のベクトルを作る。
»	定義プログラムの終わり。
SWAP	元のΣDATを1段目に移す。
STOΣ	ΣDATを以前の値に戻す。
»	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' MEDIAN STO	プログラムをMEDIANとして保存。

例　268ページのデータの中央値を計算してみましょう。(この例では268ページのデータをキー入力してあることを前提にしています。)このデータは2列なので、MEDIANは要素が2個のベクトルを返します。
中央値を計算しましょう。

USER
MEDIAN



1列目の中央値は14.5で、2列目の中央値は9.5です。

ディレクトリ移動

ここには2個のプログラムがあります。

- UPは親ディレクトリのメニューが見えるようにします。
- DOWNは子ディレクトリのメニューが見えるようにします。

このプログラムは全体のディレクトリ構造をいつでも覚えていて、どこにいるかがいつもわかっていれば不要ですが、そうでないときには、非常に便利です。

UP(親ディレクトリのどれかに動く)

親ディレクトリ、そのまた親ディレクトリの順にHOMEディレクトリまでの名称が入っているメニューを作ります。

引き数	結果
1：	1：

技法

- 親ディレクトリのリスト　UPはPATHを使ってその時点のディレクトリと全部の親ディレクトリのリストを返します。
- リストの一部取り出し　UPはSUBを使ってPATHのリストからその時点のディレクトリ名だけを取り除きます。
- ユーザ定義メニュー　UPはMENUを使って、修正したPATHリストから親ディレクトリのユーザ定義メニューを作ります。

プログラム	注釈
≪	プログラムの始まり。
PATH	スタックに経路リストを入れる。
1	スタックに1を入れる。
OVER SIZE 1 -	リストの要素数－1をスタックに入れる。
SUB	PATHのリストから最後の名称(その時点のディレクトリ)以外の全名称が入っているリストを作る。
MENU	親ディレクトリのメニューを作る。
≫	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' UP STO	プログラムをUPとして保存。

例　HOMEディレクトリから初めて、子ディレクトリD1、D2、D3を階層的に作り、UPを使ってD3からD1に動きましょう。

スタックをクリアしてHOMEディレクトリに動きましょう。

[ON] CLEAR
[MEMORY] HOME

子ディレクトリD1を作ってそこに動きます。

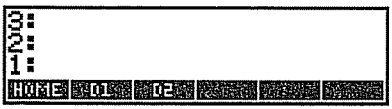
[']D1 CRDIR
D1 ENTER

子ディレクトリD2とD3についてこれを繰り返します。

[']D2 CRDIR
D2 ENTER
[']D3 CRDIR
D3 ENTER

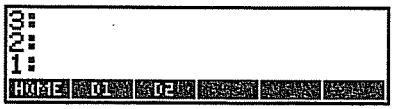
親ディレクトリのメニューを表示してみましょう。

UP ENTER



D1ディレクトリに動きましょう。

D1



DOWN(子ディレクトリのどれかに動く)

その時点の全部の子ディレクトリ名が入っているメニューを作ります。

引き数	結果
1：	1：

技法

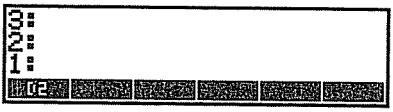
- 変数のリスト　DOWNはVARSを使ってその時点のディレクトリ内の変数と子ディレクトリのリストを返します。
- エラーを捕える　VARSリスト内の名称がディレクトリかどうかを調べるために、DOWNはRCLの引き数としてその名称を使います。ディレクトリだとスタックに呼び出すことができないので、名称がディレクトリならエラーが発生するため、その名称をディレクトリ名のリストに追加します。

プログラム	注釈
`«`	プログラムの始まり。
`VARS`	変数名と子ディレクトリ名の全部のリストをスタックに入れる。
`→ v`	VARSのリストをローカル変数vに保存。
`«`	定義プログラムの始まり。
`{ }`	ディレクトリ名を入れるリスト(最初は空)をスタックに入れる。
`1 v SIZE`	1とvの要素数をスタックに入れる。
`FOR j`	v内の各名称ごとに、次の処理をします。
`v j GET`	名称を取り出す。
`IFERR RCL DROP`	名称で変数の内容を呼び出してみる。エラーが発生しなかったらそれを捨てます。
`THEN +`	RCLでエラーが発生したら、その名称はディレクトリ名なので、ディレクトリ名のリストにその名称を追加する。
`END`	THEN文節とプログラム構造の終わり。
`NEXT`	v内の次の名称を繰り返す。
`MENU`	ディレクトリ名の専用メニューを作る。
`»`	定義プログラムの終わり。
`»`	プログラムの終わり。
ENTER	プログラムをスタックに入れる。
' DOWN STO	プログラムをDOWNとして保存。

例　277ページの前例で子ディレクトリD1、D2、D3を作って、ディレクトリD1に移ったままで終わりました。この例ではD2に移り、それからD3に移りましょう。

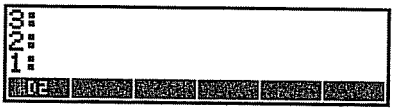
子ディレクトリのメニューを表示してみます。

DOWN ENTER



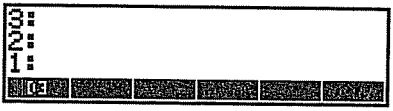
D2に下降してみましょう。

D2



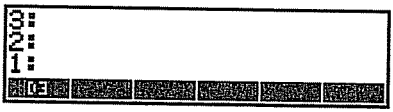
子ディレクトリのメニューを表示してみます。

DOWN ENTER



D3に移ってみましょう。

D3



付録と索引

付録A

一般的技術情報、電池、修理

ここにはHP-28Sで何か問題があったときに役立つ記事があります。HP-28S使用法で分からないことがあって、5ページの目次または332ページの索引に適切な項目が見当たらないときには、下記の「Q&A集」の部分を見てください。

電池の交換法は286ページを見てください。計算機が正しく働いてないように思われましたら、289ページの「修理が必要かどうかの判定法」を見てください。計算機の修理については、291ページの「製品の保証」と293ページの「修理が必要な場合」を見てください。

Q&A集

Q：[ON]を押しても表示画面に何も見えません。何が悪いのでしょうか？
A：これは直ちに解決できるようなこともありますし、計算機の修理が必要なこともあります。289ページの「修理が必要かどうかの判定法」を見てください。

Q：計算機が正しく働いていることを確認することができますか？
A：290ページの「反復試験(自己診断)」を試してください。

Q：計算機内のメモリ全体をクリアするにはどうしますか？
A：20ページの「全メモリのクリア(メモリのリセット)」のように[ON]と[INS]、[▶]の3キーを同時に押して放します。

Q：表示画面の右端に点が3個(...)見えるのは何でしょうか？
A：この3点を省略記号とも呼び、オブジェクトが1行では表示できないことを表します。

Q：あるオブジェクト全体を見るにはどうしますか？
A：173ページの「既存のオブジェクトの修正」の説明のように、▒ EDIT または▒ VISIT を使って、そのオブジェクト行に返します。カーソル機能キーを使うと、オブジェクトのどの部分でも見られるようになります。修正を取り消すには、ON を押します。

Q：「オブジェクト」とはどういう意味ですか？
A：「オブジェクト」とは実際に取り扱う対象全般を指す一般的用語です。数値、数式、配列などはどれもオブジェクトの1種です。オブジェクトの型の簡単な記事は25ページの「主な機能と概念」を、オブジェクトの型の詳細説明は154 ページの「オブジェクト」を見てください。

Q：計算機から音が出て、Bad Argument Type の表示になりました。何が悪いのでしょうか？
A：スタックにあるオブジェクトが、実行しようとしたコマンドに必要な正しい型でないからです。例えば、1段目に名称がないのに STO を実行するとこのエラーになります。31ページの「コマンドのカタログ」で説明したように、CATALOGを使ってそのコマンド用の正しい引き数を調べてください。

Q：計算機から音が出て、 Too Few Arguments の表示になりました。何が悪いのでしょうか？
A：スタックにあるオブジェクトが、実行しようとしたコマンドに必要なものより少なかったのです。例えば、スタックに数値が1個だけのときに + を実行するとこのエラーになります。31ページの「コマンドのカタログ」で説明したように、CATALOGを使ってそのコマンド用の正しい引き数を調べてください。

Q：計算機から音が出て、上の2件とは違うメッセージが出ました。何が悪いかをどう調べたらよいでしょうか？
A：Reference ManualのAppendix A　"Messages" を見てください。

Q：音が出ないようにするにはどうしますか？
A：51 SF とキー入力して ENTER を押します。これでフラグ51をセットするので、音が出なくなります。

Q：表示内容をハード・コピー(印字)できますか？
A：ON を押したまま L を押し、それから ON も放します。

Q：[A]から[R]のキーが働きません。何が悪いのでしょうか？
A：うっかりとメニュー優先にしてしまったようですので、■を押さないで[A]から[R]を押すと、メニューを切り換える機能にしまったのです。メニュー優先を解除するには、■[MENU]を押します。

Q：以前に使ったいくつかの変数が見つかりません。どこに行ったのでしょうか？
A：別のディレクトリでその変数を使ったのでしょう。どのディレクトリで使ったのかを覚えていなかったら、全部のディレクトリを調べる必要があります。275ページの「ディレクトリ移動」を参照にしてください。

Q：メモリの未使用量を調べる方法は？
A：MEM[ENTER]を実行すると自由に使えるメモリのバイト数が返ります。

Q：カーソルの形が変わるのはなぜですか？
A：カーソルの形でその時点の入力モードを表すからです。入力モードは普通入力(外枠だけ)、数式入力(2本線入り)、文字入力(黒)のどれかです。カーソルの輪郭で重ね書きモード(長方形)、挿入モード(矢形)を表します。172ページの「カーソルの形による入力モードの識別」を見てください。

Q：名称をキー入力した(またはUSERメニュー・キーを押した)けれども、その名称がスタックに入りません。なぜですか？
A：引用符で囲まない名称を入力したからで、これは変数の内容を指します。スタックに名称を入れるには、まず[']を押します。(57ページの「引用符で囲んだ名称と囲まない名称」を見てください。)

Q：−27の立方根を計算したときに、−3が返らないのはなぜですか？
A：どの数値にも3個の立方根があって、その内の2個は複素数です。HP-28Sは3個の立方根の中の1個を返し、これを主値と呼びます。引き数が正の実数なら、主値は実根です。引き数が負の実数なら、主値は複素根です。実数aのb番目の実根を計算するには、次のプログラムをキー入力します。

```
« → a b 'SIGN(a)*ABS(a)^INV(b)' »
```

['][RROOT][STO]と押して、このプログラムを変数RROOT(real root、実根)に保存します。これで27[CHS][ENTER]3[ENTER]RROOT[ENTER]と押すと、−27の実数の立方根を求めることができます。

Q：計算機がいつもよりも遅くなって、案内表示の  が点滅するようになりました。どうなったのでしょうか？
A：計算機がトレース印字モードになっています。[SHIFT] [PRINT] [TRACE] と押すと、トレース印字モードから抜け出します。

Q：プリンタは始めの数行は速く印字しますが、後は遅くなります。なぜですか？
A：計算機はプリンタに一定量のデータまでは速く送り、それ以後はプリンタが追従できるように伝送速度を遅くしています。

Q：印字速度を上げる方法は？
A：プリンタにACアダプタを接続してあると、計算機から高速にデータを送信してもプリンタが追従できるようになります。高速印字に切り換えるには、52 SF [ENTER] と操作します。これは印字速度を制御している、フラグ52をセットします。プリンタにACアダプタを接続しないときには、フラグ52をクリアするために 52 CF [ENTER] と操作して、普通の印字速度に戻します。

Q：プリンタの字が抜けたり、[SHIFT]の文字を印字します。何が悪いのでしょうか？
A：プリンタと計算機との距離または角度が大きいか、途中に邪魔物があるからです。プリンタと計算機との位置の詳細はプリンタのマニュアルを見てください。

Q：STOとSTOREの違いは何ですか？
A：STOコマンドは指定した値を変数に代入します。STOREメニューには、変数内の値を引き数として使って演算結果を元の変数に代入する、変数演算用のコマンドが入っています。

Q：計算結果として記号を期待したのに、数値の結果になってしまいました。なぜですか？
A：1個か2個以上の変数に値を代入してあるからです。式に使う変数の内容を削除(52ページの「変数の削除」参照)して、もう一度計算してください。

Q：[DRAW] を押したら、表示が消え、案内表示の((●))が点滅してから消えましたが、表示部には何も見えません、なぜですか？
A：計算結果がその時点のプロット範囲外です。91ページの「プロット目盛の変更」を見てください。

Q：変数または数式を評価したら、計算機が応答しなくなってしまいました。[ON]を押しても効果がありません。どうなったのでしょう。
A：変数内または式中にそれ自体の名称を定義したり、循環する定義を作ったので、計算機が無限に続くループを実行中なのです。
このループを止めるには、次のようにシステム停止を実行します。

1. [ON]を押し続けます。
2. [▲]を押す。
3. [ON]も放す。

変数または数式から循環定義などを削除して、改めて定義してください。

電池

HP-28Sは3本のアルカリ乾電池を使います。新品の電池なら平均して約半年～1年使えます。しかし、電池の寿命はHP-28Sの使用方法によって変わります。

新品の単5型アルカリ乾電池(AM5型、アメリカではN型、ヨーロッパではLR1型)を使ってください。同じ大きさのアルカリではないマンガン乾電池(UM5、R1)もありますが、こちらは寿命が半分以下です。交換用乾電池は家電製品店やカメラ店でもお求めになれますが、利用対象製品が少ないため売ってない店もありますのでお求め前にご確認ください。

低電圧表示

低電圧表示()が見えるようになると、少くとも10時間はHP-28Sを連続して使えます。最初に低電圧表示が現れたときにHP-28Sのスイッチを切っておくと、不揮発性メモリの内容は約1か月持ちます。

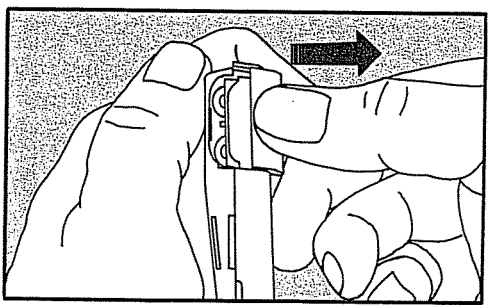
電池の交換

HP-28Sをお求めになったばかりのときには、電池は取りつけ済みなので、以下の操作を練習されると良いでしょう。

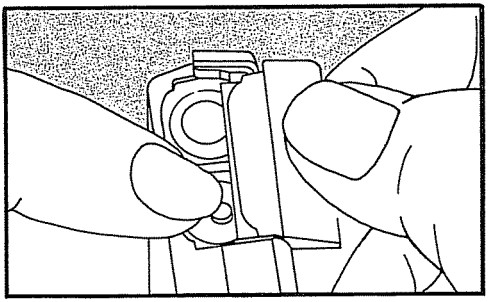
しかし、ご利用後に電池を交換するときにご注意いただきたい点は、HP-28S(不揮発性メモリ)内に記憶している情報を保存しておくためには、以下の操作に時間的制約があることです。いったん電池室のふたを取ったら、1分間以内に電池を交換してふたを閉めれば、不揮発性メモリは消えません。ですから電池室を開く前に新しい電池を用意しておいてください。それに、電池交換の間に計算機のスイッチを入れないように注意してください。

電池交換は次のようにします。

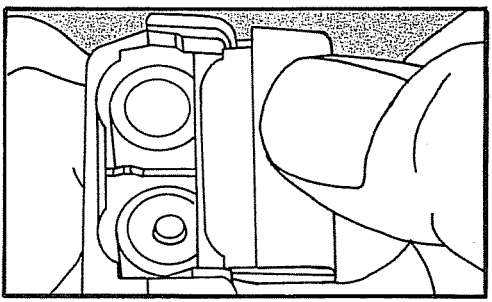
1. 新品の単5型アルカリ乾電池を机の上などに並べます。
2. HP-28Sを開いてキーボードが見えるようにします。確実に電源スイッチを切っておいてください。電池交換が完了するまでは [ON] を押さないでください。電池交換の間に電源スイッチを入れると、不揮発性メモリの内容が消えてしまいます。
3. 下図のように電池室のふたが手前側になるように持ちます。電池室のふたを取るために、計算機の裏側方向(電池の絵がある方)に引いてください。



4. 古い電池を取り出してください。
5. 新しい電池を3本入れます。電池の方向は計算機裏側の図にあります。図の通りに電池の極性(＋と－)を確認してください。
6. 指で電池を奥に押してください。



7. ふたを溝に差し込みます。必要に応じて、指で電池を奥に押してください。カチッと手ごたえがあるまでふたを横に押し込んでください。



手入れ

表示部のガラスを掃除するには、ごく小量の水で湿したティッシュ・ペーパーを使ってください。本体内に水が入らないようにしてください。

ふたの折り曲げ部分に油をささないでください。

使用環境

製品の信頼性を保つために、次の温度と湿度範囲に気を付けてください。

- 操作温度　0℃～45℃
- 保管温度　－20℃～65℃
- 操作と保管湿度　40℃でも相対湿度90%以下。

計算機の修理が必要かどうかの判定法

計算機が正しく働くかどうかを調べたいときには、次のように操作してください。計算機の修理が必要なときには、291ページの「製品の保証」と293ページの「修理が必要な場合」を見てください。

[ON]を押しても表示部に何も見えないとき

1. 表示部の濃淡を点検してみる。
   a. [ON]を押し続けます。
   b. [+]を何回か押します。
   c. [ON]も放します。
   d. 表示部にまだ何も見えなかったら、[ON]を押して、a～cまでを繰り返してみます。

2．286ページの方法で、電池を交換してみます。
3．1と2の操作でも計算機が普通に戻らなかったら、修理が必要です。291ページの「製品の保証」と293ページの「修理が必要な場合」を見てください。

表示は見えるけれども、キーを押しても変化しないとき

1．次のようにシステム停止を実行します。
   a. [ON]を押し続けます。
   b. [▲]を押します。
   c. [ON]も放します。
2．まだ計算機が応答しなかったら、次のようにメモリをリセット(不揮発性メモリのクリア)してみます。
   a. [ON]を押したままにします。
   b. [INS]と[▶]を押し続けます。
   c. [INS]と[▶]を放します。
   d. [ON]も放します。
3．1と2でも計算機が普通に戻らなかったら、修理が必要です。291ページの「製品の保証」と293ページの「修理が必要な場合」を見てください。

反復試験(自己診断)

計算機は動くけれども、正しく働いていないみたいなとき

1．プリンタを持っていたら、スイッチを入れます。計算機の試験中に修理に役立つ数値を印字します。
2．次のように繰り返し試験を開始します。
   a. [ON]を押し続けます。
   b. [◀]を押します。
   c. [ON]も放します。

この反復試験は自動的に進行します。(試験が進行しなかったら、多分 [ON] [▼] を押したのでしょう。これは工場で採用している別な試験の開始法で、キーボードの操作が必要です。この試験から抜け出すには、下記4のシステム停止を実行し、それから正しく反復試験を開始します。)

3. 現れるメッセージに気を付けます。試験で横線や縦線、空白、不規則な模様が現れ、その後で試験結果のメッセージが現れます。
   - メッセージが 0K-28S なら、計算機は試験に合格したことを表します。
   - 1 FAIL のようなメッセージなら、計算機は試験に不合格なことを表します。最初の数値は不合格の種類を表します。計算機を修理に送るときには、この数値とプリンタで印字したものも(もし、あれば)一緒に送ってください。

     キーを押して試験を中断したときには、不合格のメッセージが出ますが、これはキー操作を期待していなかったからです。この不合格メッセージは計算機に問題があることを表しません。
4. 次のようにシステム停止を実行して試験を中止します。
   a. [ON] を押し続けます。
   b. [▲] を押します。
   c. [ON] も放します。
5. 試験を中断したのでないのに、不合格メッセージになったら、計算機の修理が必要です。次の「製品の保証」と293ページの「修理が必要な場合」を見てください。

製品の保証

保証内容

当社を含むヒューレット・パッカード社グループでは、材質上や製造工程技術などの不具合について最初にお買いいただいた日から1年間の保証をいたします(ただし電池そのものと、電池が原因の故障は除外いたします)。お買い上げいただいた後で別な人に転売したり贈り物にした場合は、1年間の保証期間の残り部分が自動的に新しい所有者の保証期間になります。この保証期間内に発生した故障品を、送料元払いで当社を含むヒューレット・パッカード社グループの修理センタにお送りくだされば、当社の判断で無償で修理するか、交換いたします。(交換で同等の新型品または改良品になる場合もあ

ります。)この場合の保証期間は当初の保証期間の残り部分です。

この保証は法律上の権利を差し上げるもので、これ以外の権利について制限を加えるものではありません。

保証の適用範囲外

電池そのものと、電池が原因の故障(例えば電池の液洩れが原因のさびや腐食など)は当社を含むヒューレット・パッカード社グループでは保証いたしません。電池と電池の液洩れ保証については電池会社にご相談ください。

事故または誤使用が原因の故障は保証適用範囲外です。また当社を含むヒューレット・パッカード社グループの修理センターまたは契約修理会社以外での修理や改造が原因の故障についても保証いたしません。

前のページに記載した以外の保証はいたしません。製品の修理または交換以外の責任はご容赦願います。修理の際にお客さまが製品に記憶させておいた情報が失われることがあることをあらかじめお断りいたします。製品の販売後に改良のための仕様変更がありましても保証の範囲外とさせて頂きます。

修理が必要な場合

当社を含むヒューレット・パッカード社グループは世界の主要国に修理センタを開設しています。計算機の保証期間に関係なく、正しく動作しないときはいつでも修理または当方の判断によって交換(同等品または改良品)に応じます。ただし1年間の保証期間が切れたものは有料になります。大部分の計算機は現品到着の5作業日以内に修理して返送しています。年数が古い物や修理センタの作業状況によっては遅くなることもあります。製造打切後5年間は修理に応じることをお約束しますが、これ以降は修理用部品の在庫状況により修理に応じられないことがあります。

修理センター

- **日本**　裏表紙内側の修理センタに送ってください。お買い上げの販売店に発送を依頼することもできます。
- **アメリカ**　裏表紙内側のアメリカの修理センタに送ってください。
- **ヨーロッパ**　修理センターの所在地は、各国のヒューレット・パッカード社の事務所または販売店、あるいは次のHP社のヨーロッパ統括本部にお問い合わせください。大部分の修理センタはヒューレット・パッカード社の事務所とは別な所在地になっていますので、事前連絡なしで要修理品をヒューレット・パッカード社の事務所へ送ることはしないでください。
  Hewlett-Packard S.A.
  150, Route du Nant-d Avril
  P.O.Box CH 1217 Meyrin 2
  Geneva, Switzerland
  電話：スイス(022)82 81 11
- **その他の国**　修理センタの所在地は、各国のヒューレット・パッカード社の事務所または販売店、あるいは裏表紙内側のアメリカの修理センタにお問い合わせください。修理センタがない国では、要修理品をアメリカの修理センタに送ってくださっても結構です。
  要修理品の修理センタへの荷造・送料、修理完成品の輸入手続き、税関の通関料などはお客様に負担していただくことになっています。

修理料金

保証期間が切れた製品の修理料金は機種ごとの標準料金制です。この標準料金は裏表紙内側の修理センタにおたずねください。一部の国または州ではこの標準料金に付加価値税または取引税などの税金を加算した金額になります。事故または誤使用によって破損した製品の修理料金は標準料金制ではありません。この場合は部品代と修理工数によって個々に算出いたします(日本では修理着手前にお客様の同意を得た上で修理を開始いたします)。

発送方法

要修理品は各国の修理センタ(国または地方によっては配送センタ)に送ってください。(修理センタへの送料は、保証の有無に関係なくお客様がご負担ください。)次のことを確認してください。

- 修理完成品の返送先住所とお名前、故障の症状などを明記した紙をいれてください(日本以外では担当修理センタ所在国の公用語で書いてください)。
- 保証期間内であれば、お買い上げ日を証明する書類を入れてください(当社の保証書はアメリカなどの外国でも通用いたします。)
- 日本以外の修理センタに修理を依頼する場合には、標準修理料金(一部の国や州では税金を加算した金額)を記入し署名したその担当修理センタで通用する銀行小切手、あるいはVISAまたはMasterCardのクレジット・カードのカード番号とカード有効期限を書いて署名した紙を入れてください。(ただしYHPではクレジット・カード利用の取り扱いをしておりません。)日本やアメリカなど一部の修理センタでは、有料修理品で修理料金を前もって入れていただいていない場合は、修理完成時に代金引換郵便または代金着払い便で発送いたします。
- 要修理品と添付書類は輸送途中で破損しないように適切な荷造りをしてください。輸送途中での破損と紛失は保証の対象になりませんので、書留郵便または輸送保険のご利用をお勧めいたします。
- 修理センタまでの送料などは保証の有無に関係なくお客様がご負担ください。

有償修理品の修理後の保証

有償修理品は修理に使用した部品と修理技術に対して90日間の保証をいたします。

安全と規制についての情報

受信障害規制

アメリカ　HP-28Sはコンピュータと同様に高周波を発生させているので、ラジオやテレビの受信を妨害する可能制があります。しかしHP-28SはアメリカのFCC(電波管理委員会)の一般住宅での受信障害発生防止を目的としている受信障害防止規制(Subpart J of Part 15 of FCC Rules)で規定するClass B計算装置の規制試験で問題がないことを確認してあります。もしもテレビやラジオの受信障害が発生したら(これはHP-28Sの電源スイッチを入・切するか電池を抜いてみれば分かります)、次のようにして障害を除いてください。

- 受信アンテナの向きを変えてみる。
- 受信機への方向や距離を変えてみる。

必要に応じて、製品販売店や熟練したラジオ・テレビ技術者に相談してください。アメリカ国内では次の英文文書を入手して読むのも一つの方法です。

編 著 者　Federl Communication Commission
書　　名　How to Identifiy and Resolve Radio-TV Interference Problems
発 行 元　U.S.Goverment Printing Office
住　　所　Washington D.C.20402
整理番号　Stock number 004-000-00345-4
電　　話　アメリカ　(202)783-3238

西ドイツ　HP-28SとHP 82240Aプリンタは西ドイツのVFG 1046／84とVDE 0871Bや類似の障害防止基準に従っています。

ヒューレット・パッカード社が承認していない製品を使う時には、その組み合わせが西ドイツの1984年12月14日付官報VFG 1046／84の第2節に従う試験などが必要です。

付録B

HP社製RPN 計算機に慣れている方への注意事項

1972年のHP-35以来、ヒューレット・パッカード社では逆ポーランド法のスタックを使った科学技術用と金融計算用のハンドヘルド型計算機を各種開発してきました。計算機能や計算用途の違いはありましたが、どれも基本のスタックの使用法は同じで、どれか一つの計算機になれると残りの機種の使用法を学習するのは簡単でした。

HP-28Sもユーザと対話する部分にはスタックと逆ポーランド法を使っています。しかし、従来の計算機の4段スタックと固定式のレジスタではHP-28Sの各種オブジェクト型を扱ったり代数式の計算機能の実現には不適当でした。そこでHP-28Sでは本来の逆ポーランド法を大きく発展させました。HP-28Sと従来の機種との間に大きな違いがあって、従来の機種になれている人でも使いこなすための予備知識が必要です。ここでは、大きな違いの要点を説明することにします。

動的なスタック

HP-28Sと従来のHP社製RPN 計算機を比較したときに一番大きく違う点はスタックの段数です。以前の機種ではX、Y、Z、Tレジスタで構成した固定式の4段スタックにLAST XまたはLレジスタを1個追加したものでした。スタックをクリアしたり、全部に0を入れたりしても、このスタックはいつも存在していました。

HP-28Sではスタックの段数は固定ではありません。スタックに新しいオブジェクトを入れるたびに、必要なだけ新しい段を自動的に作っています。スタックからオブジェクトを取り出すと、スタックが低くなって、スタックがなくなることさえもあります。そこでHP-28Sでは従来のHP社製RPN 計算機にはなかった Too Few Arguments(引き数が不足)エラーが発生することもあります。

この動的スタックと固定スタックとの違いでHP-28Sと従来の機種との間には特に次のような違いがあります。

番号付きのスタック段　HP-28Sのスタックの段数は一定でないためにX、Y、Z、Tというスタックの段名は不適当になったので、各段に番号を付けました。そこで1段目はXレジスタ、2はY、3はZ、4はTとほとんど同じです。わかりやすいようにキー記号の$1/x$とx^2はHP-28Sにも残してあります(この方がそれぞれのコマンド名のINV とSQよりもキー記号として視覚的にわかりやすいでしょう)。しかし、RPN に定着しているX<>YはHP-28CではSWAPと改名しました。

スタック操作　HP-28Sでは固定スタック式計算機よりもスタック操作のコマンドが多数必要です。例えば、R↑とR↓はそれぞれROLLとROLLDに置き替えましたが、これにはスタックを何段回転させるかを指定する引き数が余分に必要になりました。STACKメニューには固定スタック式計算機にはなかったようなスタック操作コマンドもいくつか入っています。

Tレジスタの自動コピーの廃止　固定スタック式計算機では、スタック降下(つまり、スタックから数値が1個減ったとき)のたびにTレジスタの内容をZレジスタに自動的にコピーしていました。これは一定の数値の掛け算を繰り返すのに便利な方法で、定数をスタック全体に入れておいてから、被乗数を入力して[×]を押した結果を記録しては[CLx]を押すという操作を何回でも繰り返すことができました。HP-28Sではこの方法は使えませんが、次の形のようなプログラムは簡単に作れます。プログラム中の12345 は定数の例です。

```
« 12345 * » 'MULT' STO
```

この操作がすんでから[USER]を押します。数値を入力してから [MULT] を押し、新しい数値を入力してから [MULT] をまた押し、以下同様にすることで定数掛け算ができます。この方法ではスタックに結果が連続して残ります。

メモリとしてのスタック　動的スタックには計算に必要なだけ、始めの方のスタックが蒸発してしまうという心配もなく、何段でも使えるという長所があります。これは計算が終わった後に結果をスタックに残しておくと、不要なオブジェクトでかなりのメモリ量を使うという短所にもなります。HP-28Sでは、不要になったオブジェクトはスタックからどんどん捨ててください。

DROP対CLX　固定スタック式計算機では、CLXは「Xレジスタの内容を0に書き替えて、その次にスタック上昇が発生しないようにする(下記参照)」という意味です。これの主な用途は、新しい数値を入れる前に古い数値を捨てることで、これで0を入れたものとしてそのまま使うこともできました。HP-28Sでは、CLXはDROPに代わりました。その名前のようにスタックの1段目のオブジェクトを捨てて、スタック内の残りのオブジェクトをどれも1段ずつ下に移動させます。無関係な0が入ることはありません。同様に、固定式スタック全体を0に置き替えるCLST(CLEAR STACK)の代わりに、CLEARはスタックから全オブジェクトを捨ててしまいます。

スタック上昇防止とENTER

固定式スタック計算の一部のコマンド(ENTER ↑、CLX 、Σ+、Σ−)はスタック上昇と呼ぶ独特の機能が働かないようにしていました。つまり、このコマンドのどれかを実行した後に、スタックに新しい数値を入れるとXレジスタの内容を書き替えるだけで、Xレジスタの内容をYレジスタに押し上げることはありませんでした。この機能はHP-28Sでは完全になくなりました。スタックに新しいオブジェクトを入れるとスタックのそれ以前のオブジェクト全体が必ず1段ずつ上に上がります。

固定式スタック計算機のXレジスタとENTER はスタック構造というよりも表示部が1行分しかないために2種の役目を持っていました。Xレジスタは普通のスタック・レジスタであると同時に入力レジスタを兼ねていて、数値を入力するときには、数値入力キー以外で入力を打ち切るまでは、Xレジスタに1桁ずつ数値が入っていきました。[ENTER ↑]キーは前と後の数値を区切る役目でした。しかし数値入力を区切る以外に、[ENTER ↑]キーはXレジスタの内容をYレジスタにコピーして、その後の入力によるスタック上昇を防止していました。

HP-28Sではこの二つの役目を分離して、スタック上昇防止はなくなりました。コマンド行がスタックの1段目(前のXレジスタ)から完全に分かれて、コマンドやオブジェクトの入力に使うようになりました。ENTERはコマンド行の内容の処理に使うだけで、スタックの1段目の内容をコピーすることはありません。ただし、コマンド行がないときに ENTER キーを押すとDUP(1段目の内容を2段目にコピー)を実行するので注意してください。この ENTER の機能は従来の計算機との類似性として残したものです。

前置きと後置きとの比較

HP-28Sのコマンドは完全な後置記法です。つまり、引き数を使うコマンドはどれでもコマンドを実行する前にその引き数がすでにスタックにあることが必要です。これは従来のRPN 計算機とは違っていて、従来はレジスタ番号、フラグ番号などを指定する引き数をスタックに入れないで、コマンドそのものの後に入れていました(例えば、STO 25、TONE 1、CF 03などです)。この従来の方法はスタックを1段使わなくてもすむという長所もありますが、指定方法に融通性がないという短所もありました。例えば、HP-41ではSTOの後にレジスタ番号の2桁がいつも必要です。

HP-28Sの操作に似ているのは固定式スタック計算機で間接指定する方法で、ここではコマンドで指定したいレジスタやフラグ番号などの入れ物として *i* レジスタ(HP-41では任意のレジスタ)を使っていました。HP-28SではSTOやRCLなどでスタックの1段目を *i* レジスタのように使っていると考えてください。例えば、RCLは「1段目にある名称の変数(従来のレジスタ)の内容を呼び出す」という意味でHP-41のRCL IND X と同じことです。

HP-28Sの大部分のコマンドで注意することはスタックからその引き数を取り去ってしまうことです。例えば、123 ' X ' STOを実行すると、123と ' X ' はスタックから消えてしまいます。この働きがないと、スタックには古い引き数がどんどん増えてしまいます。スタックに123を残しておきたかったら、123 DUP ' X ' STOと実行することが必要です。

レジスタと変数との比較

固定式スタック計算機では実数の浮動小数点数だけを扱っていたので、スタックには固定で7バイト長のレジスタ構造と番号付きのデータ・レジスタ・メモリが適していました(HP-41で7バイト以内の原始的な文字列オブジェクトが扱えるようになりました)。HP-28Sでは番号付きデータ・レジスタの代わりに名称付きの変数を使っています。変数は各種のオブジェクト型に合うような柔軟な構造になっているだけでなく、レジスタ番号よりもその内容を速く思い出すことができるような名称を付けることができます。

HP-28Sで番号付きレジスタを再現するには、ベクトルを使います。

```
{ 50 } 0 CON 'REG' STO
```

これで要素が50個で初期値が0のREGという名のベクトルを作ったことになります。

```
« 'REG' SWAP GET » 'NRCL' STO
```

これでこのベクトルからn番目の要素(このnはスタックの1段目に入れる数値)を呼び出すNRCLというプログラムができあがります。

```
« 'REG' SWAP ROT PUT » 'NSTO' STO
```

これで同様にベクトルに保存するプログラムNSTOができあがります。

LASTX とLASTとの比較

固定式スタック計算機ではLASTXコマンドはLASTX(またはL)レジスタの内容を返しました。このレジスタはXレジスタで最後に使った数値のコピーを入れておくところです。HP-28SではLASTコマンドでこの概念をもっと一般化して、コマンドがスタックから取り出した1ないし3個の引き数を返します(4個以上の引き数をスタックから取り出して使ってしまうコマンドはありません)。そこで固定式スタック計算機の１２ ＋ LASTXではスタックに3と2が返りますが、HP-28Sの１２ ＋ LASTではスタックに3と1、2が返ります。

HP-28SのLASTは従来のLASTXよりもずっと柔軟ですが、HP-28Sではスタックから引き数を取り出すコマンドが固定式スタック計算機よりも多いことに注意してください。これはLASTの引き数がもっと頻繁に更新されてしまうことを意味し、DROPやROLLのようなコマンドでもLASTの引き数を書き替えてしまいます。

UNDOはスタック全体を書き替えるので、簡単なエラー回復にはLASTよりも具合がよいことがあることを忘れないでください。

付録C

算式通り入力の計算機に慣れている方への注意事項

単純な加減乗除ていどの四則計算機を含めて、大部分の計算機には各種の算式通り入力の操作方式があります。この名称は簡単な計算のときに使うキー操作順序が、本などに印刷されている算式にほぼ似ていることから来ています。そこで、1＋2－3を評価するには、 [1] [+] [2] [−] [3] [=] と押します。

この方式は簡単な算式では＋や－、×、÷のような演算記号がその引き数の数値と数値の間に入るのでうまく働きます。少し高級な計算機では優先順位(計算の順序)の指定に括弧を使うこともできます。しかし、SIN やLOG などのような演算記号が前にくる関数が入ってくると2種の方法に分かれます。

- 普通の算式通り入力の計算機では、加減乗除のような数値間演算記号は中間のままにして、前付き演算記号はRPN計算機のように後から入力する、という2種の方法を組み合わせています。例えば、1＋SIN(23)は [1] [+] [2] [3] [SIN] [=] と操作します。この方式には中間結果を見ることができることと、前付き演算記号でも1回のキー操作だけで良い(つまり、演算記号の後に括弧が不要である)という利点がありますが、普通の関数式などになると算式通りという主な長所がなくなってしまうのが短所です。

- 「直接算式入力」の計算機とBASIC 言語のコンピュータには直接実行状態(ダイレクト・モードなどと呼んでいて、プログラム入力でない状態)があって、普通の算式通りに算式をキー入力することができ、入力終了キー(機種によってキー記号が違っていて [ENTER]、[ENDLINE]、[RETURN] など)を押すとただちに結果を算出します。この方法の長所は算式とキー操作が対応していることですが、短所は中間結果が見られないことです。(HP-71BのCALC状態は唯一の例外です。)算式の入力を始める前にその算式全体を知っておくことが必要なので、中間結果を確めながら状況に応じた計算をするのは困難です。

HP-28Sに慣れる方法

HP-28Sの操作手法はポーランドの論理学者のヤン・ウカシェーヴィッチ(Jan Łukasiewicz、1878～1956)が考案した「ポーランド記法」という数学記法に基づいています。一般の算式記法は数式を評価するときに関連する数値や変数の間に演算記号を置いています。ウカシェーヴィッチの記法は演算記号を数値や変数の前に指定します。この方法を変形したものが数値や変数の後に演算記号を指定する方法で、これを「逆ポーランド記法(略して逆ポーランド)」と呼び〝RPN〟と略記しています。

RPNの基本的な考え方は、まず数値または他のオブジェクトを入力し、次に入力してあるもの(これを「引き数」と呼びます)を使うコマンドなどを実行することです。「スタック」は後で使うオブジェクトを順々に入れておくところです。大部分のコマンドはその結果をスタックに返し、それ以後のコマンドはこれを引き数として使えます。

HP-28Sは各種の数学機能をどれも同じ方法で実現できるようにRPNのスタックを使っています。算式では表すことができないような制御操作も含めて、どんな操作でも必要な引き数をスタックから取り出して、その結果をスタックに返すという同じ方法で実行しています。

そうは言っても、算式通り入力の計算機を使ってきた人がRPN 計算機の使い方を学ぶ上での最大の障害は単純な計算にもRPN のスタックを使うということのようです。RPN は非常に効率的ですが、計算を始める前に算式を頭の中で並べ替えることが必要になります。しかしHP-28Sには数式を人間が翻訳しなくても解釈する機能があるので、従来のRPN計算機よりも数式通りの計算機にぐっと近づいたことになります。さらに、4行の表示部で4段までのスタック内容を一度に見ることができるので、スタックに迷う心配がなくなりました。

数式を評価するという目的から見ると、HP-28Sは実質的には「算式通りに入れる」計算機です。つまり、算式を評価するには、まず最初に ['] キーを押し、次に算式通りに中間演算記号や前置き関数、括弧も含めてキー入力して、そして [EVAL] を押すと計算結果が出ます。この方法は四則演算にも使うことができます。

['] [1] [+] [2] [−] [3] [EVAL] は 0 を返します。

最初の ['] 以外は、これは単純な算式通りの計算機のキー操作順序と同じで、この [EVAL] は [=] の代わりです。

注　記

HP-28Sの [=] キーと算式通り入力の計算機の [=] キーとを混同しないでください。HP-28Sの [=] は等式を作るときにだけ使います(Reference Manualの〝ALGEBRA〟に説明があります)。

HP-28Sを「算式通りに入力する計算機」として使うと、計算した各結果がスタックに順々に入って、「計算順の結果スタック」の役目になります。これで後に古い結果をまた使うために保存しておくことができるわけです。これは複雑な長い計算式を小さな短い計算式に分解して、各部分の結果をスタックに入れておいて、全部が揃ってからそれを組み合わせて結果にすることもできます。(これを徹底したのが、RPN算法の基本です。)この計算順のスタックがあるので、算式通り入力またはBASIC計算機のような結果だけが求まる機能よりも使いやすくて強力になっているのです。

HP-28Sで重要なことは、RPNの考え方が算式通り入力の考え方より秀れているか劣っているかを気にしなくてよいことです。解くべき問題に応じて適した方法を選んだり、RPNに算式の操作方法を混ぜてもよいのです。

付録D

メニュー案内

この付録部分にはHP-28Sの各メニュー内のコマンドの表があります。このメニューはALGEBRA からTRIGまで、ABC順に並べました。各メニューの詳細はReference ManualのDictionaryを見てください。このDictionaryも全部のメニューがABC順になっています。特定のコマンドの詳細は、Reference Manual巻末のOperation Indexをまず見てください。このIndexは全部のコマンドをABC順に並べてあって、そのコマンドの説明があるページがわかります。

この付録にはCATALOG、FORM、SOLVER、UNITSで利用できる対話型操作のメニューはありません。

- CATALOGは22章に記事があり、31ページに使用例があります。
- FORMは112ページの「FORMの使用法」に記事があります。詳細はReference Manualの〝ALGEBRA(FORM)〟を見てください。
- SOLVERは第8章の「SOLVER」に記事があります。詳細はReference Manualの〝SOLVE〟を見てください。
- UNITSは141ページの「UNITSのカタログ機能」に記事があります。詳細はReference Manualの〝UNITS〟を見てください。

この付録の各メニューでは、表示部に同時にみえる6個以内のコマンドごとに行で分けました。 NEXT を押すと次の行に動き、▒ PREV を押すと以前の行に動きます。

「コマンド」の欄が表示部に見える名称です。「記事」欄がそのコマンドの働きです。「ページ」欄は本書にそのコマンドの例、説明、記事などがあるページです。ここに数字がないコマンドは、Reference Manualの〝Operation Index〟を見てください。

ALGEBRA（代数）

	コマンド	記　　事	ページ
行1	COLCT	同類項をまとめる	111
	EXPAN	式の展開	111
	SIZE	オブジェクトの個数	
	FORM	数式の整理(数式エディタ)	112
	OBSUB	オブジェクトの置き替え	
	E&SUB	式の置き替え	
NEXT			
行2	TAYLR	ティラー級数	
	ISOL	変数の分離	112
	QUAD	2次式の根	
	SHOW	変数の明示	
	OBGET	オブジェクトの取り出し	
	E&GET	式の取り出し	

ARRAY（配列）

	コマンド	記　　事	ページ
行1	→ARRY	スタックから配列に	275
	ARRY→	配列からスタックに	274
	PUT	要素を入れる	
	GET	要素を取り出す	
	PUT+	要素を入れて添字を増やす	
	GET+	要素を取り出して添字を増やす	
NEXT			
行2	SIZE	要素数	274
	RDM	大きさの変更	
	TRN	転置	264
	CON	定数行列	
	IDN	単位行列	
	RSD	残差	
NEXT			
行3	CROSS	外積	126
	DOT	内積	126
	DET	行列式	128
	ABS	フロベニウスのノルム	
	RNRM	行のノルム	
	CNRM	列のノルム	
NEXT			
行4	→R→C	実数から複素数に	
	C→R	複素数から実数に	
	RE	実数部	
	IM	虚数部	
	CONJ	共役	
	NEG	正負符号反転	

BINARY(ビット整数)

	コマンド	記　　事	ページ
行1	DEC	10進数表示	140
	HEX	16進数表示	139
	OCT	8 進数表示	140
	BIN	2 進数表示	140
	STWS	ワード長保存	139
	RCWS	ワード長呼び出し	
NEXT			
行2	RL	左回転	
	RR	右回転	
	RLB	1バイト左回転	
	RRB	1バイト右回転	
	R→B	実数からビット整数に	261
	B→R	ビット整数から実数に	
NEXT			
行3	SL	左に1ビット移動	
	SR	右に1ビット移動	
	SLB	左に1バイト移動	
	SRB	右に1バイト移動	
	ASR	算術的に右1ビット移動	
NEXT			
行4	AND	論理積	
	OR	論理和	
	XOR	排他的論理和	
	NOT	否定	

COMPLEX（複素数）

	コマンド	記　　事	ページ
行1	R▶C	実数から複素数に	83
	C▶R	複素数から実数に	83
	RE	実数部	83
	IM	虚数部	84
	CONJ	共役	84
	SIGN	単位ベクトル	84
NEXT			
行2	R▶P	直交座標から極座標に	86
	P▶R	極座標から直交座標に	85
	ABS	絶対値	85
	NEG	正負符号反転	85
	ARG	偏角	85

LIST(リスト)

	コマンド	記　　事	ページ
行1	→LIST	スタックからリストに	181
	LIST→	リストからスタックに	181
	PUT	要素を入れる	271
	GET	要素を取り出す	237
	PUTI	要素を入れて添字を増やす	271
	GETI	要素を取り出して添字を増やす	271
NEXT			
行2	POS	位置	237
	SUB	一部取り出し	276
	SIZE	要素数	271

LOGS(対数)

	コマンド	記　　事	ページ
行1	LOG	常用対数	78
	10^X	逆常用対数	78
	LN	自然対数	78
	e^X	自然対数の逆関数	78
	LNP1	1＋Xの常用対数	78
	EXPM1	EXP-1	78
NEXT			
行2	SINH	双曲線正弦	78
	ASINH	逆双曲線正弦	78
	COSH	双曲線余弦	78
	ACOSH	逆双曲線余弦	78
	TANH	双曲線正接	78
	ATANH	逆双曲線正接	78

MEMORY(メモリ)

	コマンド	記　　事	ページ
行1	MEM	使用可能なメモリ量	188
	MENU	ユーザ定義メニューを作る	195
	ORDER	変数の順序を変える	184
	PATH	その時点の経路	67
	HOME	HOMEディレクトリに切り替える	71
	CRDIR	子ディレクトリを作る	66
NEXT			
行2	VARS	その時点のディレクトリ内の変数	184
	PURGE	その時点のディレクトリ内をクリア	184
	[illegible]		
	[illegible]		
	[illegible]		
	[illegible]		

MODE

	コマンド	記　　事	ページ
行1	STD	標準の数値表示形式	38
	FIX	固定小数点表示形式	38
	SCI	指数部付表示形式	38
	ENG	工学式指数部付表示形式	38
	DEG	度単位	74
	RAD	ラジアン単位	74
NEXT			
行2	CMD	COMMANDが働くようにするか働かないようにする	210
	UNDO	UNDOが働くようにするか働かないようにする	211
	LAST	LASTが働くようにするか働かないようにする	211
	ML	複数行表示にしたり、1行表示に戻す	208
	RDX	小数点をコンマにしたり、点に戻す	37
	PRMD	モードを表示して印字	

PLOT(プロット)

	コマンド	記　　事	ページ
行1	STEQ	解析用数式の保存	90
	REQ	解析用数式の呼び出し	
	PMIN	左下隅座標指定	95
	PMAX	右上隅座標指定	95
	INDEP	独立変数指定	
	DRAW	プロット表示	90
NEXT			
行2	PPAR	プロット・パラメータの呼び出し	90
	RES	解像度	
	AXES	座標軸の交点指定	
	CENTR	画面中心座標指定	94
	XFAC	表示横倍率変更	
	YFAC	表示縦倍率変更	93
NEXT			
行3	STOS	統計用行列に保存	
	RCLS	統計用行列の呼び出し	
	COL	独立変数と従属変数の列を指定	
	SCLS	散布図表示範囲の自動設定	
	DRAW	散布図の表示	
	[illegible]		
NEXT			
行4	CLRGT	表示画面のクリア	
	DGTIZ	座標読み取り	
	PIXEL	指定座標に点を打つ	
	DRAX	座標軸を表示	
	NORMP	普通表示に戻す	
	PRLCD	表示画面を印字	

PRIN(印字)

	コマンド	記　　事	ページ
行1	PRI	1段目を印字	151
	PRSTK	スタックを印字	152
	PRVAR	変数名とその内容を印字	152
	PRLCD	表示画面を印字	149
	CR	プリンタ・ヘッドを改行	
	TRAC	トレース・モードにするか抜け出す	150
NEXT			
行2	PRSTC	スタックを印字	
	PRUSR	ユーザ変数名を印字	
	PRMD	モードを印字	

PROGRAM BRANCH(プログラムの分岐)

	コマンド	記　　事	ページ
行1	IF	IF文節の始まり	226
	IFERR	IF ERROR文節の始まり	227
	THEN	THEN文節の始まり	226
	ELSE	ELSE文節の始まり	226
	END	プログラム構造の終わり	226
NEXT			
行2	START	指定回数ループの始まり	228
	FOR	指定回数ループの始まり	229
	NEXT	指定回数ループの終わり	228
	STEP	指定回数ループの終わり	230
	IFT	IF THENコマンド	227
	IFTE	IF THEN ELSE関数	226
NEXT			
行3	DO	条件停止ループの定義	231
	UNTIL	条件停止ループの定義	231
	END	プログラム構造の終わり	231
	WHILE	条件停止ループの定義	232
	REPEAT	条件停止ループの定義	232
	END	プログラム構造の終わり	232

PROGRAM CONTROL(プログラム制御)

	コマンド	記　　事	ページ
行1	SSTEP	1ステップずつの実行	250
	HALT	プログラム実行を止める	234
	ABORT	プログラム実行を中断	
	KILL	停止しているプログラムを中断	250
	WAIT	プログラムの一時停止	234
	KEY$	キー文字列を返す	234
NEXT			
行2	BEEP	指定した音を出す	234
	CLEAR	表示画面をクリア	234
	DISP	指定行に表示	234
	CONT	普通の表示に戻す	234
	ERRN	エラー番号の呼び出し	
	ERRM$	エラー・メッセージの呼び出し	

PROGRAM TEST(プログラム中の試験)

	コマンド	記　　事	ページ
行1	SF	フラグをセット	205
	CF	フラグをクリア	205
	FS?	フラグをセットしてあるか？	225
	FC?	フラグをクリアしてあるか？	
	FS?C	フラグをセットしてあるか調べてクリア	
	FC?C	フラグをクリアしてあるか調べてクリア	
NEXT			
行2	AND	論理積	
	OR	論理和	
	XOR	排他的論理和	
	NOT	否定	232
	SAME	同一	231
	==	等価	222
NEXT			
行3	STOF	フラグを保存	156
	RCLF	フラグの呼び出し	156
	TYPE	オブジェクトの型	232

REAL(実数)

	コマンド	記　　事	ページ
行1	NEG	正負符号反転	78
	FACT	階乗(ガンマ関数)	78
	RAND	乱数	78
	RDZ	乱数の種を指定	78
	MAXR	最大の実数	79
	MINR	最小の実数	79
NEXT			
行2	ABS	絶対値	
	SIGN	正負符号	
	MANT	仮数部	
	XPON	指数部	
NEXT			
行3	IP	整数部分	
	FP	小数部分	
	FLOOR	下側の整数に切り捨て	272
	CEIL	上側の整数に切り上げ	272
	RND	四捨五入	
NEXT			
行4	MAX	最大値	
	MIN	最小値	
	MOD	剰余	
	%T	全体比	

SOLVE(ソルブ)

	コマンド	記　　事	ページ
行1	STEQ	式の記憶	64
	RCEQ	式の呼び出し	
	SOLVR	SOLVER の変数メニュー	102
	ISOL	変数の分離	110
	QUAD	2 次式の根	108
	STOV	変数の明示	
NEXT			
行2	ROOT	根の算出	

STACK(スタック)

	コマンド	記　　事	ページ
行1	DUP	複製	178
	OVER	2番目のオブジェクトを複製	178
	DUP2	2個のオブジェクトを複製	178
	DROP2	2個のオブジェクトを捨てる	179
	ROT	回転	178
	LIST→	リストからスタックに	181
NEXT			
行2	ROLLD	下方に移動	178
	PICK	取り出し	178
	DUPN	n個のオブジェクトの複製	178
	DROPN	n個のオブジェクトを捨てる	179
	DEPTH	スタックの高さ	181
	→LIST	スタックからリストに	181

STAT(統計)

	コマンド	記　　事	ページ
行1	Σ+	統計用行列にデータを入れる	132
	Σ−	統計用行列からデータを取り出す	133
	NΣ	統計用行列の行数	134
	CLΣ	統計用行列をクリア	132
	STOΣ	統計用行列に保存	275
	RCLΣ	統計用行列を呼び出す	264
NEXT			
行2	TOT	統計用行列内の列和	
	MEAN	統計用行列内の平均	134
	SDEV	統計用行列内の標準偏差	135
	VAR	統計用行列内の分散	135
	MAXΣ	統計用行列内の最大値	
	MINΣ	統計用行列内の最小値	
NEXT			
行3	COLΣ	独立変数と従属変数の指定	136
	CORR	相関係数	136
	COV	共分散	136
	LR	直線回帰の係数	137
	PREDY	予想値	137
NEXT			
行4	UTPC	χ^2分布の上側確率	
	UTPF	F分布の上側確率	
	UTPN	正規分布の上側確率	
	UTPT	t分布の上側確率	
	COMB	組み合わせ	
	PERM	順列	

STORE(保存)

	コマンド	記　　事	ページ
行1	STO+	和を保存	
	STO-	差を保存	
	STO*	積を保存	
	STO/	商を保存	
	STNEG	正負反転値を保存	
	STINV	逆数を保存	
NEXT			
行2	STOΣ	共役値を保存	
	[illegible]		
	[illegible]		
	[illegible]		
	[illegible]		
	[illegible]		

STRING(文字列)

	コマンド	記　　事	ページ
行1	→STR	オブジェクトを文字列に	258
	STR→	文字列をオブジェクトに	175
	CHR	文字コードを文字列に	156
	NUM	文字列を文字コードに	156
	→LCD	グラフィクス文字列を画面に再生	157
	LCD→	画面をグラフィクス文字列に	157
NEXT			
行2	POS	位置	
	SUB	一部取り出し	
	SIZE	文字数	258
	DISP	指定行に表示	156

TRIG(三角関数)

	コマンド	記　　事	ページ
行1	SIN	正弦	74
	ASIN	逆正弦	74
	COS	余弦	74
	ACOS	逆余弦	74
	TAN	正接	74
	ATAN	逆正接	74
NEXT			
行2	P→R	極座標を直交座標に	76
	R→P	直交座標を極座標に	76
	R→C	実数を複素数に	76
	C→R	複素数を実数に	76
	ARG	偏角	76
NEXT			
行3	→HMS	10進数を60進数に	76
	HMS→	60進数を10進数に	76
	HMS+	60進数の和	76
	HMS-	60進数の差	76
	D→R	度をラジアンに	77
	R→D	ラジアンを度に	77

キーの索引

この索引はキーボードの各キーの働きをまとめたものです。最初は左側のキーボードのキーをABC 順に並べ、次に右側のキーボードのキーをABC 順に並べました。最後はカーソル機能です(これは右側の最上行のキーの上側に白で印刷してあります)。

この索引には■ [ARRAY]や■ [OFF]のようにシフト・キーを併用するものも入れてあります。[A]から[Z]と[0]から[9]の文字キーは、いつでもコマンド行に1文字書き込むだけなので、索引には入れてありません。([(]のような識別記号、[=]のような演算記号、■ [π]のような記号定数のようにこれ以外の文字も入れてありません。このような文字は計算機にとっては特別な意味がありますが、それぞれのキーは単なる文字キーです。)この索引にないキーは、文字キーです。

各キーごとに、その働きの要点と参照ページを載せました。本書にキーの説明がないとか、もっと詳しい情報がほしいときには、Reference Manualの巻末のOperation Index を見てください。

左側のキーボード

右側のキーボード

カーソル機能の表示

カーソル機能の表示はメニュー・キー(右側のキーボードの最上行)の上側に白で印刷してあります。カーソル機能はコマンド行があって、メニュー記号が表示されてないときに使えます。メニュー記号が表示されているときにカーソル機能に切り替えるには、[◆▶]を押します。以前のメニュー記号に戻るには、もう一度[◆▶]を押します。

キー	記　事	ページ
[INS]	重ね書きモードと挿入モードの切り替え	167
■[INS]	カーソル位置の左側の全文字を削除	168
[DEL]	カーソル位置の文字を削除	167
■[DEL]	カーソル位置とその右側の全文字を削除	168
[▲]	カーソルを上に動かす	167
■[▲]	カーソルを最上行に動かす	168
[▼]	カーソルを下に動かす	167
■[▼]	カーソルを最下行に動かす	168
[◀]	カーソルを左に動かす	167
■[◀]	カーソルを左端に動かす	168
[▶]	カーソルを右に動かす	167
■[▶]	カーソルを右端に動かす	168

記事の索引

太字の数字はまず見ていただきたいページ、普通の数字はその次に見ていただきたいページです。

【カ行】

【サ行】

【タ行】

【ナ行】

【ハ行】

【マ行】

【ヤ行】

【ラ行】

【ワ行】

日本の修理センタ

〒229　神奈川県相模原市矢部１—27—15
横河・ヒューレット・パッカード株式会社
相模原事業所　サービスセンター
電話　0427-59-1311(大代表)

アメリカの修理センタ

Hewlett-Packard
Calculator Service Center
1030 N.E. Circle Blvd.
Corvallis, Oregon 97330, U.S.A.
Telephone: (503) 757-2002

横河・ヒューレット・パッカード株式会社

本　　社　〒168　東京都杉並区高井戸東3－29－21
営業本部　　電話　03－331－6111（大代表）

No. 00028-90125
Printed in Japan 12/88