ＡＰＬＥＸＬを用いてＥＸＣＥＬ上で使用するＡＰＬ

Ａ）　ＡＰＬ計算式の基本

Ｂ）　ＡＰＬＥＸＬプログラムの操作方法

Ｃ）　さまざまな計算

Ｃ－１．集計と一般的計算

Ｃ－２．時間、カレンダー、年齢などに関する計算

Ｃ－３．金利に関する計算

Ｃ－４．漢字の処理とＣＳＶファイルの入出力について

Ｄ）　その他のＡＰＬの主な機能

Ｄ－１．算術演算子の例（“ＡＰＬ２の世界”―著者編集（無償配布）より）

Ｄ－２．その他の論理、比較演算子

Ｄ－３．三角関数

Ｄ－４．作用子の役割

２００６年3月１０日

ＡＰＬコンサルタンツ　三枝　協亮

A）　ＡＰＬ計算式の基本

ＡＰＬの計算式は関数を中心として次の６通りの形があります。

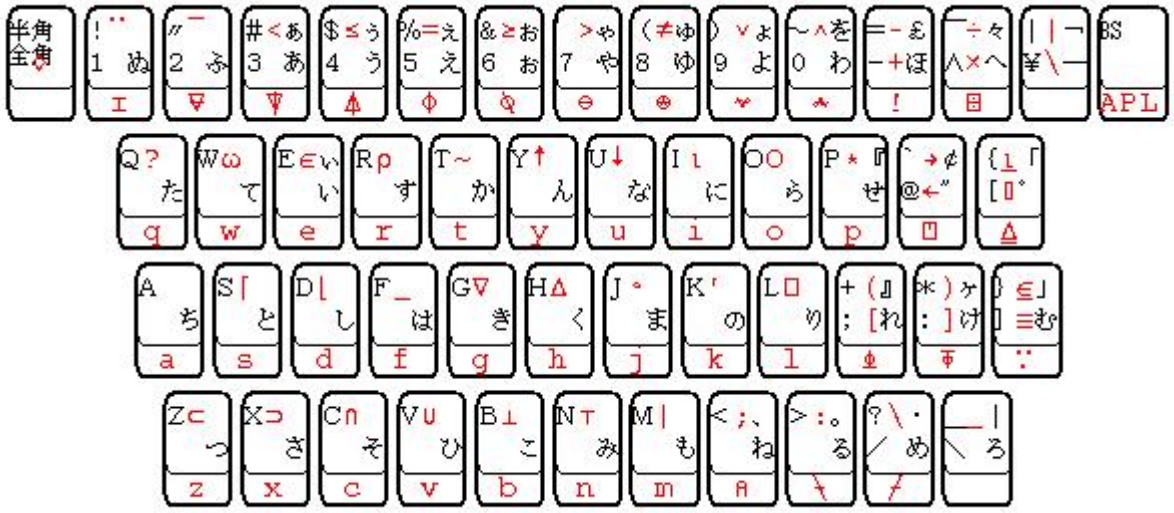
	結果値を持たないもの	結果値を持つもの
引数を持たないもの(Niladic)	関数	結果値←関数
右引数だけを持つもの(Monadic)	関数　引数	結果値←関数　引数
左右に引数を持つもの(Dyadic)	引数　関数　引数	結果値←引数　関数　引数

関数にはシステムに付随した原始関数（特殊文字１字で表される)、システム関数（第１字が□文字のもの)、外部関数（結合プロセサーを経由して使用できるもの）とユーザーがプログラムを作り定義する定義関数などがあります。また関数を修飾して派生関数を作る作用子があり、これにもシステムに付随した原始作用子とユーザーが定義する定義作用子があります。

関数　｜－　原始関数　　（ＡＰＬ特殊文字１字からなる名前を持つ）：[シ]
　　｜－　システム関数（□文字で始まる名前を持つ）　　：[シ]
　　｜－　外部関数　　（作業領域外に存在し、結合宣言で使用可能になる）：[シ]、[ユ]
　　｜－　派生関数　　（関数を作用子で修飾して生成される）　　：[ユ」
　　｜－　定義関数　　（＝ＡＰＬプログラム：ユーザー指定の名前を持つ）：[ユ]
　註：[シ] ＝システム提供関数、[ユ] ＝ユーザー定義関数

原始関数には Monadic が２８個、Dyadic が４５個あり、それぞれＡＰＬ固有の特殊文字１字で表されます。どちらでも使用されている文字がありますので両方で合計５０個の特殊文字があります。ＡＰＬ特殊文字を画面上に表示したり、印刷出力するためには、ＩＢＭ　ＡＰＬ２製品版（有料）または　Ｒｕｎｔｉｍｅ　Ｍｏｄｕｌｅ（無料）が正しくＷｉｎｄｏｗｓに導入されている必要があります。(http://aplcons.com　からダウンロード可)

下の図は鍵盤上のＡＰＬ文字配列を示しています。現在日本語仕様のＡＰＬ鍵盤は市販されていませんが、ＡＰＬ文字は一般の日本語鍵盤から入力することができます。

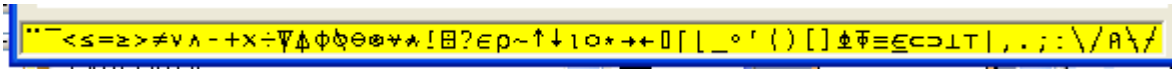


ＡＰＬ文字は鍵盤がＡＰＬシフトの状態のときに入力できます。ＡＰＬとＡＳＣＩＩシフトの

切替えホット・キーはＢＳ＋Ｃｔｒｌですが、ＡＰＬＥＸＬではカーソルの位置によって自動的に切り替えを行っていますから、ユーザーが操作する必要はありません。

ＡＰＬシフトの状態ではキーの上面に表示された文字のうち、ＡＳＣＩＩ特殊文字とひらがな（含む漢字）は入力できません。数字と英字の大文字、およびキー面上に２段で表示された赤い字の下側の文字が入力されます。Ｓｈｉｆｔ・キーを押した状態では、キー面上の上側に表示された赤い文字が入力されます。Ａｌｔキーを押した状態ではキーの前面の赤い字（英字小文字とＡＰＬ特殊文字の一部）が入力されます。

鍵盤からのＡＰＬ文字入力に慣れるまでは、入力効率は落ちますが、入力画面の下方のＡＰＬ文字入力補助域からマウスクリックで入力することができます。隠れた部分には大小英字および数字も含まれていますから、左右にスクロールするか、またはウインドウズを右に拡大（推奨）すれば表示された文字はすべて入力できます。



定義関数はユーザーが定義するプログラムですが、他の関数と同様、上に示した６種類のどれかの形に属します。また関数ですから、作用子により修飾して派生関数とすることもできます。

特定の値で代表される引数（変数、定数、計算の結果値など）は必ず次の属性値を持ちます。

値（ＶＡＬＵＥ）：数値はほぼ±１．７９の１０の３０８乗、文字は任意のバイト、又は無。
形（ＳＨＡＰＥ）：軸の本数（ＲＡＮＫ、最大６３）とそれぞれの長さ（ＬＥＮＧＴＨ）
領域（ＤＯＭＡＩＮ）：数値領域と文字領域
深さ（ＤＥＰＴＨ）：配列を構成要素とした配列のネスティング最大１８１階層
名前（ＮＡＭＥ）：第１字アルファベット、最長２５５英数字、または無名（定数など）

結果値を持つ関数は実行の結果、値を作り出しますから値そのものとみなすことができます。したがって他の関数の引数として指定することができます。そのような関数は１行のＡＰＬ計算式のどの部分にでも組み込むことができます。たとえば下に示す構成が成り立ちます。

（引数　関数　引数）　関数　（引数　関数　（関数　引数））

＝引数（３番目に実行される）　＝引数（１番目に実行される）

＝引数（２番目に実行される）

＝主関数（最後に実行される）

ＡＰＬの計算式が右側から実行されるというのは正しい表現ではなく、正しくは、関数の実行に先立って、引数が用意されなければならないということです。たとえば１０×５－３において、関数×の左引数は１０ですが、右引数は５－３です。したがって、５－３の結果値が求まらなければ関数×は実行することができません。関数－の左引数は５で右引数は３ですから引数が全部そろっていますので、実行することができます。すなわち１０×５－３は１０×２で、答えは２０で、４７であるためには（１０×５）－３でなければなりません。

ただし関数が左右の引数をとる（＝二項形式）場合、右引数が左引数より先に解釈されます。これは次の実験で証明することができます。(註：◇は関数でなく、同一行中の複数の独立した式を区切る区切り文字です。)

```
      A←5 ◇ (A←10)+A
15
      A
10
```

すなわち、変数Ａに５を代入し、Ａに１０を代入する左引数と右引数のＡを関数＋で計算すると答えが、１５になります。もし左引数が優先するなら答えは２０とならなければなりません。

また、括弧は実行の順序を制御するのに用いられますが、上の式では不必要な括弧が用いられています。下のようにすれば式の表現はすっきりします。

（引数　関数　引数）関数　引数　関数　関数　引数

括弧をすべて取り除くと一番左の関数がその右の計算結果を右引数として最後に実行される関数になります。どのような表現も成り立ちますが、関数の並べ方で、括弧の少ないもの、求めたいものが（文章のように）明確に表現できたものが優れたＡＰＬの式の立て方といえます。

引数　関数　引数　関数　引数　関数　関数　引数
―――――――――――――――――――――――――
＝右引数

ＡＰＬでは、算術演算、論理演算、三角関数演算、そのほかすべての関数は同列に扱われ、論理的に成り立つものであれば、どのような組み合わせでも許されます。ＡＰＬＥＸＬではＥｘｃｅｌワークブックのシートをＸ１，Ｘ２，Ｘ３，・・・と、その物理的な順番に従って名前をつけ、ｎ行，ｍ列の２次元配列として使用します。配列の大きさは可変で、使用されるセルが必ず含まれるサイズがダイナミックに確保されます。

１つのセルは配列の１要素（＝スカラー）で、文字データであればそれを関数ＤＩＳＣＬＯＳＥ（⊃）で開くか、またはＥＮＬＩＳＴ（∈）で階層構造を壊せば、文字配列であり、１文字ずつを要素（＝スカラー）とした配列です。したがって、ＡＰＬから見れば１枚のＳＨＥＥＴは、２階層（Ｄｅｐｔｈ＝２）の入れ子(ネスティング)構造の文字と数値要素からなる複合配列(＝Ｃｏｍｐｌｅｘ　Ａｒｒａｙ)です。ただし数値セルだけを取り出した場合は、各セルは配列としての構造を持ちませんから、１階層（Ｄｅｐｔｈ＝１）の単純配列(＝Ｓｉｍｐｌｅ　Ａｒｒａｙ)になり、取り扱いが異なる場合があります。(註：スカラーとは構造を持たない（＝ＳＨＡＰＥが無とみなされる）配列です。)

たとえば配列ＡとＢの各セル同士が等しいか、否かを判定する場合、通常Ａ≡¨Ｂと、原始関数ＭＡＴＣＨ≡を原始作用子ＥＡＣＨ¨で修飾した派生関数を用います。Ａ≡Ｂだけなら、全体の比較になりますので、答えは１か０ひとつしかありません。¨をつけると、セルそれぞれについての比較になりますから、答えもＡ，Ｂと同じサイズの１／０の配列になります。ただし数値配列の場合は、原始関数＝を用いてもかまいません。何故なら＝は他の多くの演算関数と同様、対象配列の各要素（＝スカラー）に対して作用する（＝Ｐｅｒｖａｓｉｖｅ）スカラー関数だからです。ＡＰＬの大部分の原始関数はスカラー関数です。

ＡＰＬの原始関数、原始作用子、システム関数、システム変数、外部関数のすべてを以下表記しますので、参考にしてください。システム関数、システム変数、外部関数の一部はＡＰＬのシステム環境に依存しますので（例：作業空間関連）ＡＰＬＥＸＣＬでは使用できません。原始関数と原始作用子はすべて使用できます。外部関数はあらかじめＡＰＬＥＸＬに組み込まれたもの（名前の頭に△をつけました）以外は、必要に応じて、その使用を次のような式で結合（＝ＡＳＳＯＣＩＡＴＩＯＮ）宣言をします：

システム提供のものに関しては：

　　Ｔ←３　１１　□ＮＡ　‘外部関数名’　（註：Ｔは結果値、０＝成功、０≠失敗）
または
　　Ｔ←３　１１　□ＮＡ　‘代替名　外部関数名’

ユーザー定義関数（＝プログラム）に関しては：

　　Ｔ←‘外部関数ライブラリー名’　１１　□ＮＡ　‘外部関数名’

外部関数はＡＰＬ言語使用の中で、比較的新しいコンセプトで、次のような特徴があります：

１．作業領域外のライブラリーに存在し、実行され、ユーザーがみだりに変更できない。
２．ユーザーがＡＰＬ言語で定義して、指定したライブラリーに追加することができる。
３．複数のユーザーが同時に利用することができる。
４．異なるライブラリー間で、関数を結合させることができる。
５．ＡＰＬ　Ｒｕｎ－ｔｉｍｅ　Ｍｏｄｕｌｅからも利用できる。

以上のような特長によって、ＡＰＬ外部関数はＡＰＬ環境下での汎用ユーティリティー関数とか､適用業務プログラムなどを複数のユーザーに対して一元管理するのに理想的な仕組みです。

原始関数シンボルの名前とバレンス（項数）

シンボル	一項使用	二項使用
+	Conjugate（共役）	Add（加算）
−	Negative（逆符号）	Subtract（減算）
×	Direction（方向）	Multiply（乗算）
÷	Reciprocal（逆数）	Divide（除算）
*	Exponential（指数関数）	Power（累乗）
|	Magnitude（絶対値）	Residue（剰余）
⌈	Ceiling（天井）	Maximum（最大）
⌊	Floor（床）	Minimum（最小）
⍟	Natural Logarithm（自然対数）	Logarithm（一般対数）
?	Roll（乱数）	Deal（抜取り乱数）
!	Factorial（階乗）	Binomial（２項係数）
○	Pi Times（円周率倍）	Circle Functions（円関数）
⌹	Matrix Inverse（逆行列）	Matrix Divide（行列除算）
<		Less Than（小さい）
≤		Less Than or Equal（小さいか等しい）
=		Equal（等しい）
≥		Greater Than or Equal（大きいか等しい）
>		Greater Than（大きい）
≠		Not Equal（等しくない）
∧		And（論理積）
∨		Or（論理和）
⍲		Nand（否定論理積）
⍱		Nor（否定論理和）
~	Not（否定）	Without（除外）
∈	Depth（深さ）	Match（均等）
,	Ravel（ベクトル化）	Catinate（連結）
⍴	Shape（形）	Reshape（変形）
⍳	Disclose（開示）	Pick（ピック）
⊂	Enclose（囲み）	Partition（区画）
⌽ ⊖	Reverse（逆順）	Rotate（回転）
⍉	Transpose（逆軸転置）	Transpose（一般転置）
↓		Drop（落し）
↑	First（最初）	Take（取り）
[]		Bracket Index（大カッコ指標）
⌷		Index（指標）
⍋	Grade Up（昇順）	Grade Up - Collating Sequence（照合順序昇順）
⍒	Grade Down（降順）	Grade Down - Collating Sequence（照合順序降順）
⍳	Interval（インターバル）	Index of（指標調べ）
∈		Find（検出）
∈	Enlist（エンリスト）	Member（メンバー）
⍎	Execute（実行）	
⍕	Default Format（省略値書式化）	Format by Example/Specification（例／指定による書式化）
⊥		Decode（復号）
T		Encode（符号化）
/		Compress（圧縮）, Replicate（反復）
\		Expand（拡張）

原始作用子

[X]	Axis（軸）
¨	Each（個別）
/	Reduce（低減）
\	Scan（走査）
.	Outer product（外積）
∘.	Inner product(内積)

システム関数

制御及び実行

□DL	Delay（遅延）
□EX	Expunge（消去）
□NA	Name Association（名前の関連付け）

事象（Event)処理及びデバッグ

□EA	Execute Alternate（代替実行）
□EC	Execute Controlled(制御付実行)
□ES	Event Simulation（事象擬似検証）

作業空間及びシステム情報

□AF	Atomic Function（原子関数）
□AT	Attributes(属性)
□NC	Name Class（名前の類別）
□NL	Name List（名前の表）
□UCS	Universal Character Set（ＵＣＳ）

データと式の変換

□CR	Character Representation（文字表現）
□FX	Fix（関数化）
□TF	Transfer Format(転送形式)

共用：

□SVO	Shared Variable Offer
□SVR	Shared Variable Retraction
□SVC	Shared Variable Control（共用変数制御）
□SVQ	Shared Variable Query
□SVS	Shared Variable State

システム変数

□	Evaluated Input/Output（解釈を伴う入出力）
□	Character Input/Output（文字入出力）
□LX	Latent Expression（潜在式）
□NLT	National Language Translation（国語変換）
□PR	Prompt Replacement（プロンプト置き換え）
□PW	Print Width（印刷幅）

□EM	Error Message(事象メッセージ)
□ET	Event Type（事象タイプ）
□LC	Line Count(行番号)

□AI	Account Information(会計情報)
□AV	Atomic Vector(原子ベクトル)
□TC	Terminal Control（端末制御文字）
□TS	Time Stamp（時刻表示）
□TZ	Time Zone（時間帯）
□WA	Work Area Available（使用可能な作業空間）

暗黙の引数

□CT	Comparison Tolerance（比較許容値）
□FC	Format Control（書式制御）
□IO	Index Origin（指標原点）
□PP	Print Precision（印刷精度）
□RL	Random Link（乱数リンク）

□SVE	Shared Variable Event(共用変数事象)

ＡＰＬ２外部関数

関連付けプロセサー１０：

ΔF 関数：テキスト・ファイルを文字列として読み書きする。
ΔFV 関数：テキスト・ファイルを CR/LF を区切りとし、文字列のベクトルとして読み書きする。
ΔFM 関数：テキスト・ファイルを CR/LF を区切りとし、二次元文字配列として読み書きする。

関連付けプロセサー１１：

ATR 関数：(Array To Record)ＡＰＬの文字配列を多言語などで要求される形に変換する。
ATS 関数：(Array To SCAR)ＡＰＬ配列をＳＣＡＲ（Slf-Contained Array）に変換する。
BEEP 関数：(BEEP)ブザーを鳴らす。
CHECK ERROR 関数：(Check system error)システム・エラーの文章を表示する。
CNS 関数：(Create Name Space)ワークスペスからネームスペースを作る。
COM 関数：(Component Object Model)Microsoft COMとのインターフェース。
CTK 関数：(Character To Kanji)Unicode漢字をShift JISに変換する。（Windowsのみ）
CTN 関数：(Character To Numeric)数字配列を数値配列に変換する。
CTUTF 関数：(Character To UTF)ＡＰＬ２文字配列をUTF-7およびUTF-8に変換する。
DIR 関数：(Directory)ディレクトリーをリストする。
DISPLAY,DISPLAYC,DISPLAYC 関数：(Display APL2 array structure)= 1 DISPLAY
ERASE 関数：(ERASE file)ファイルを削除する。
EXP 関数：(Execute In Previous namespace)現在ネームスペースをひとつ手前に戻す。
FILE 関数：(File)テキスト・ファイルを文字ベクトルとして読み書きする。
FSTAT 関数：(File Status)指定したシステム・ファイルのファイル情報を取り出す。
GETENV 関数：(Get Environment Variable)環境変数の値を問い合わせる。
HOST 関数：(Host command)ＨＯＳＴシステムのコマンドを発行する。
IDIOMS 関数：(Idioms)APL慣用句を表示する。= 1 IDIOMS
KTC 関数：(Kanji To Character)Shift JIS漢字をUnicodeに変換する。（Windowsのみ）
LIBS 関数：(Libraries)ＡＰＬライブラリー定義を表示する。
LTM 関数：(Tol To Matrix)TolリストをＡＰＬ２配列に変換する。
MD5 関数：(MD5 encoding)ＡＰＬ文字ベクトルをMD5で暗号化する。
MKDIR 関数：(Make directory)ディレクトリーを作成する。
MTL 関数：(Matrix To Tol)ＡＰＬ２配列をTolリストに変換する。
OPTION 関数：(Option)セッション・オプションの問い合わせと設定。
PFA 関数：(Pattern Fom Array)ＡＰＬ２配列の正規パターン記述を見つける。
PIPE 関数：(Pipe)システム・コマンドの方向変換。
PRINTWSG 関数：(Print workspace with GUI)ワークスペースを印刷する。
QNS 関数：(Query NameSoope)現在ネームスコープを問い合わせる。
RENAME 関数：(Rename file)ファイルの名前を変更する。
RF 関数：(Row Find)２個の配列の間で第二配列の各行の第一配列との一致する行を見つける。
RMDIR 関数：(Remove directory)ディレクトリーの削除。
RTA 関数：(Record To Array)正規パターン記述でＡＰＬ２ベクトルを配列化する。
SIZEOF 関数：(Size Of array)外部に渡すＡＰＬ２の配列のサイズを見つける。
STA 関数：(SCAR To Array)ＳＣＡＲ（Slf-Contained Array）をＡＰＬ配列に変換する。
TCL 関数：(Tool Command Language)Tolコマンドを実行する。
TIME 関数：(Time)関数の実行時間の測定。= 1 TIME
UNZIP 関数：(Unzip)対象の解凍
UNZIPWS 関数：(Unzip workspace)ワークスペースの解凍。
WSCOMP 関数：(Workspace compare)ワークスペースの比較。= 1 WSCOMP
WSCOMP_ANALYSIS:(Workspace Compare & Analysis)ワークスペース比較分析。
UTFTC 関数：(UTF To Character)UTF-7およびUTF-8をＡＰＬ２文字配列に変換する。
ZIP 関数：(Zip)対象の圧縮。
ZIPWS 関数：(Zip workspace)ワークスペースの圧縮。

関連付けプロセサー１２：

テキスト・ファイルを指定した変数名に関連づける機能。

ＡＰＬは非常に理屈っぽいプログラミング言語です。しかし一般のユーザーは常に理詰めでプログラムを組み立てているわけではありません。ルールが少なく、単純であり、禁止条項や例外がほとんどありませんから、問題に対する解法を記憶でなく、類推によって組み立て、試行テストを繰り返し、短時間に目的を達成します。そのためにＡＰＬは高級プログラミング言語でありながら、Ｅｘｃｅｌのように、一つ一つの操作の結果を確認しながら作業を進めることのできる対話式モードでも、決められた処理を一括して実行するバッチ・モードのどちらでも使用できます。

ＡＰＬがエンドユーザーに最も向いた言語である所以は、このような使用形態から、会得したものが経験として積み重なり、外国語の習得のように、生涯役に立つ個人的道具となり得るからです。また理屈はわからなくても、仲間や先輩の見様見まねで目的を達成する場面も多くあります。理屈はその経験を後から自分で納得し、長く定着させるための役割を果たします。

ＡＰＬＥＸＬはＩＢＭのソフトウエア製品である“Ｗｏｒｋｓｔａｔｉｏｎ　ＡＰＬ２”の最新機能を利用してＡＰＬコンサルタンツ社が実験的に開発した無償配布のソフトウエアで、ＡＰＬの特性をなるべく生かすように工夫してあります。

ＡＰＬＥＸＬ利用の前提

１．Ｗｉｎｄｏｗｓ　ＸＰ／２０００／２００３サーバー

２．Ｍｉｃｒｏｓｏｆｔ　Ｅｘｃｅｌ

３．ＩＢＭ　ＡＰＬ２　製品版（有料）またはＲｕｎｔｉｍｅ(無料)

４．ＡＰＬ２学習支援システム – ディレクトリー共用（共に無料）

ＩＢＭ　ＡＰＬ２　ＲｕｎｔｉｍｅとＡＰＬＥＸＬ導入の手続き

１．http://aplcons.com にアクセスして、ダウンロードをクリックする。

２．ＡＰＬＥＸＬ導入の指示に従って操作する。

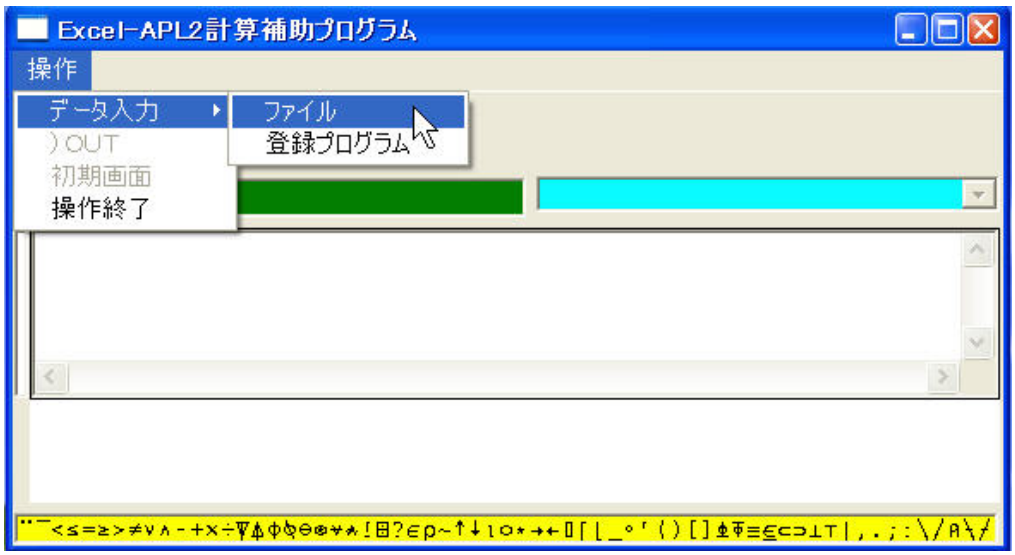
補足：ＩＢＭ　Ｗｏｒｋｓｔａｔｉｏｎ　ＡＰＬ２（Ｗｉｎｄｏｗｓ版）で使用される方は
ＡＰＬ２　Ｒｕｎｔｉｍｅの機能がすでに導入はされていますので、導入手続きが異なります。
またサーバー上での導入を考えていらっしゃる方は、kyosuke.saigusa@nifty.com にメールしてください。

B）ＡＰＬＥＸＬプログラムの操作方法

Ｂ－１．プログラムを開始する

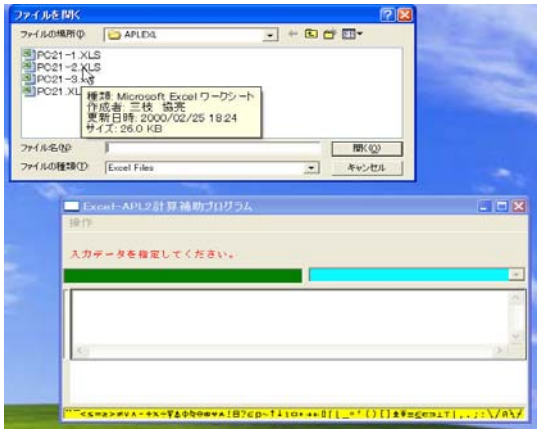
アイコンをクリックする（デスクトップ）

Ｂ－２．表示されたＡＰＬＥＸＬ画面のメニューからデータ入力を指定する

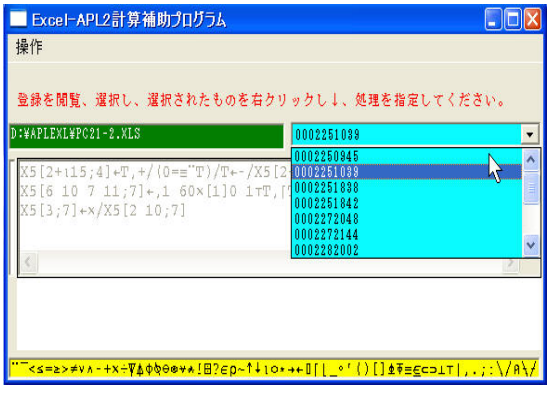


Ｂ－３．次のオプションを指定する

ファイルの場合はファイルメニュー

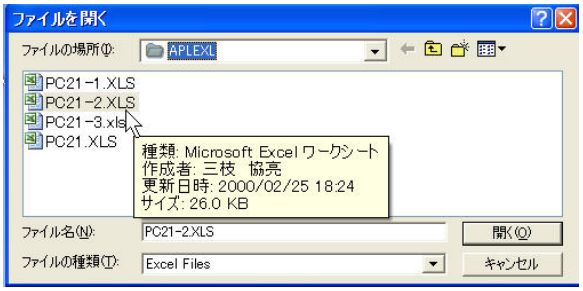


登録プログラムの場合は登録リスト

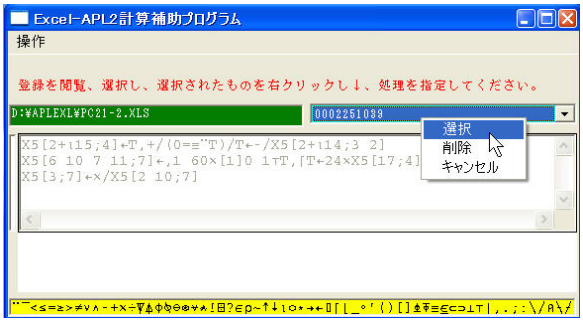


Ｂ－４．またオプションを指定する

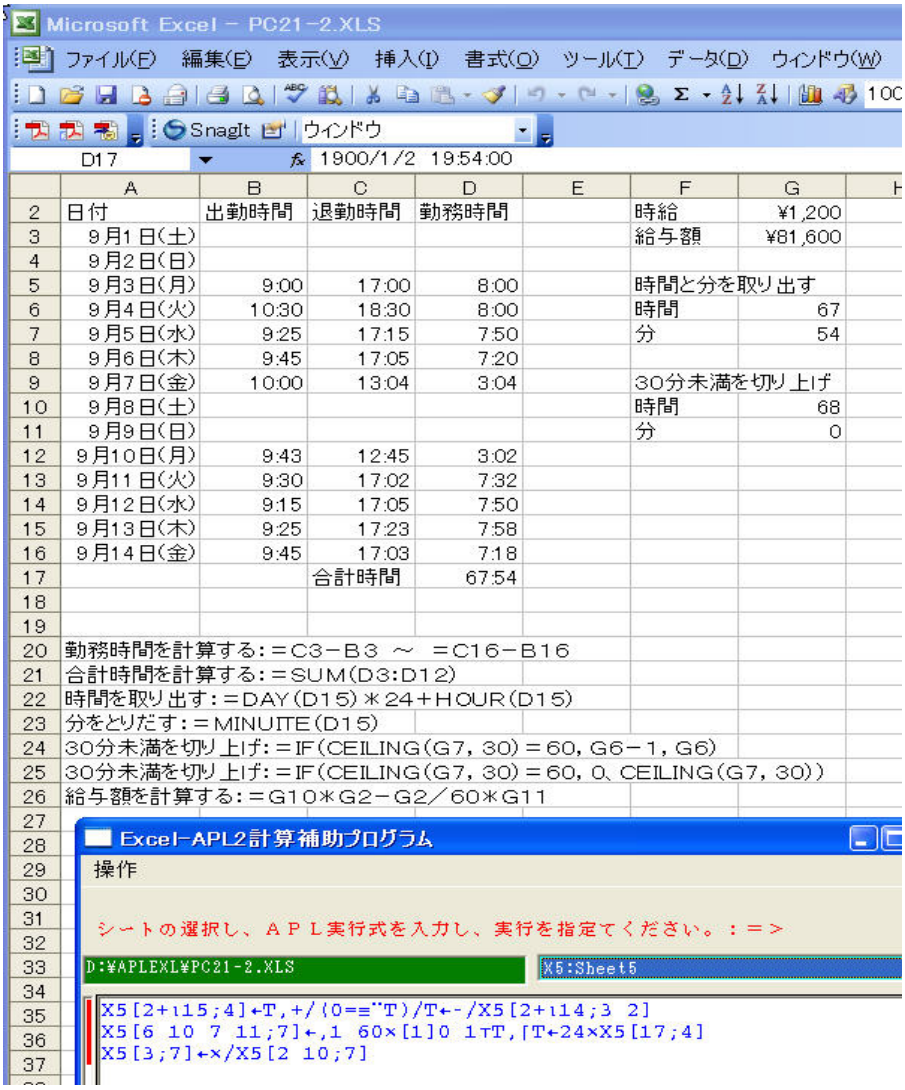
対象ファイルを選択する



右クリックのメニューでプログラム選択



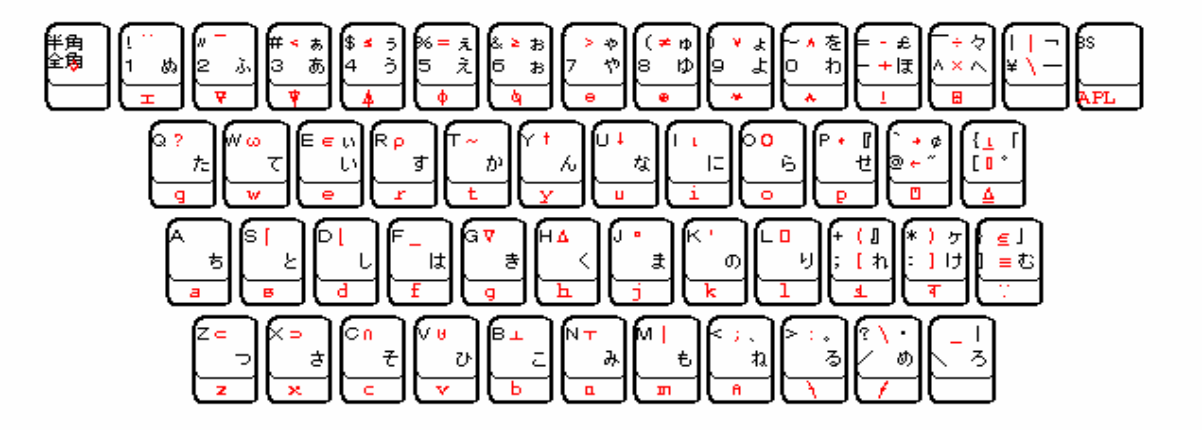
Ｂ－５．対象ブックの全ＳＨＥＥＴが後ろから順次表示される



ＡＰＬプログラムの実行を指定するまでは、Ｅｘｃｅｌの操作で自由に画面を変更することができます。

Ｂ－６．対象ＳＨＥＥＴを選び、プログラム（＝ＡＰＬ実行式）を入力または編集する

カーソルがプログラム入力域に位置するときは、鍵盤はＡＰＬシフトに設定されています。

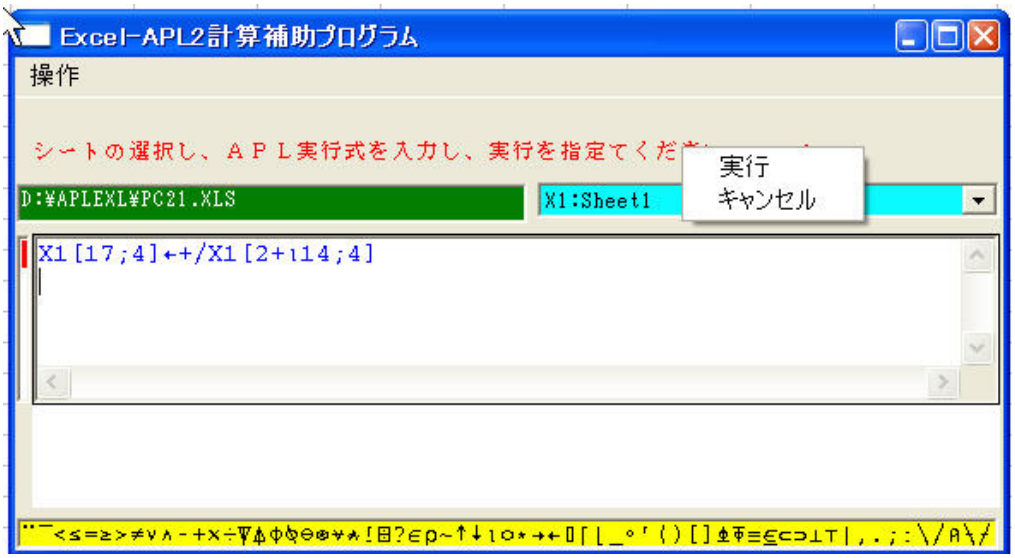


ＡＰＬシフト時、英字は大文字シフトになっています。キー面上に赤字が上下２個並んでいる場合はその下側の文字がその左の黒字文字の代わりに入力されます。たとえばマイナス（－）キーではその右隣のプラス（＋）の文字が入力されます。英字小文字を含むキー前面の文字の入力はＡｌｔキーを併用する。キー面上の赤字の文字(２個上下にある場合はその上側)の入力にはシフトキーを用います。

APL 鍵盤配列に慣れない場合は、計算補助画面下部の文字バー上の該当文字をマウスでクリックすると、カーソル位置にその文字が入力されます。表示されていない部分の文字はこの文字バーを左右にスクロールか、画面をマウス操作で横に広げることができます。

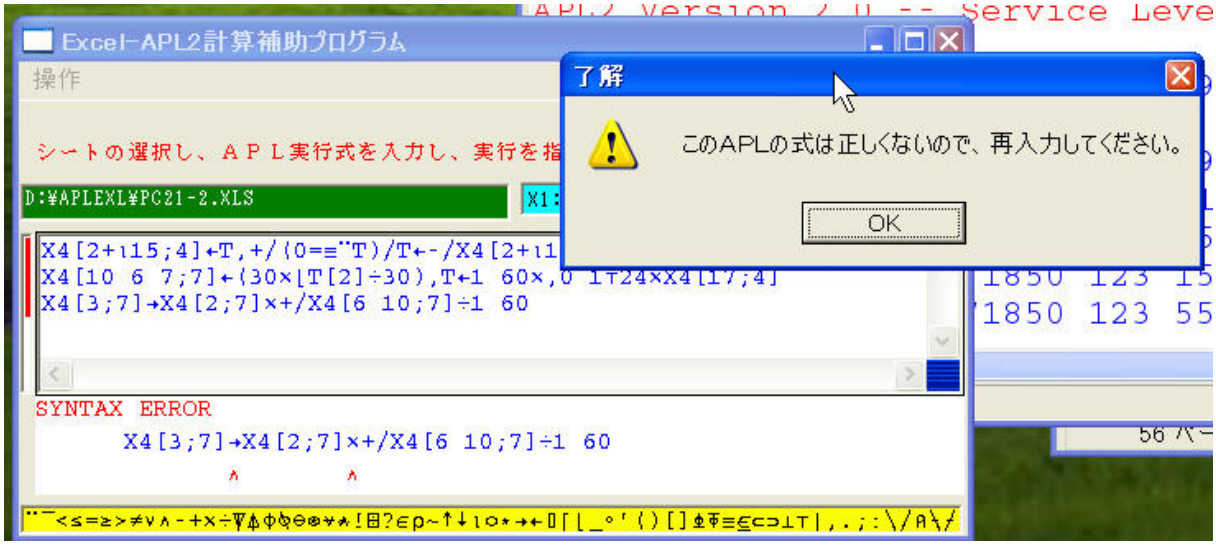
Ｂ－７．プログラムを実行する

プログラムを入力(または修正)後、画面上部で右クリックし、ポップアップメニューから実行を指示します。



Ｂ－８．エラー表示時の対応

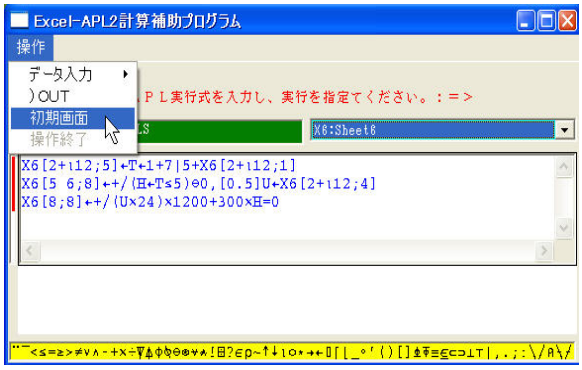
プログラム記述に誤りがあると、入力画面の下にエラー情報が表示され、確認を要求されます。

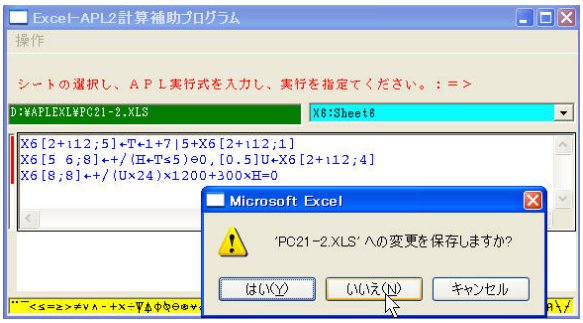


了解すると、入力画面から誤りの修正ができる。修正、Ｅｘｃｅｌ操作、実行は正しい処理が完了し、Ｅｘｃｅｌ画面上に結果が表示されるまで、何度でも繰り返すことができます。

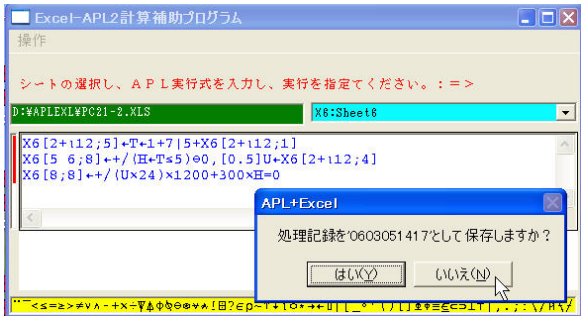
Ｂ－９．操作の終了または中断

操作を終了、中断する場合は操作メニューで指示する。画面右上のＸ印の操作は無効。
正しく実行されたプログラムの場合、Ｅｘｃｅｌファイルの更新が必要か否か要求されます。





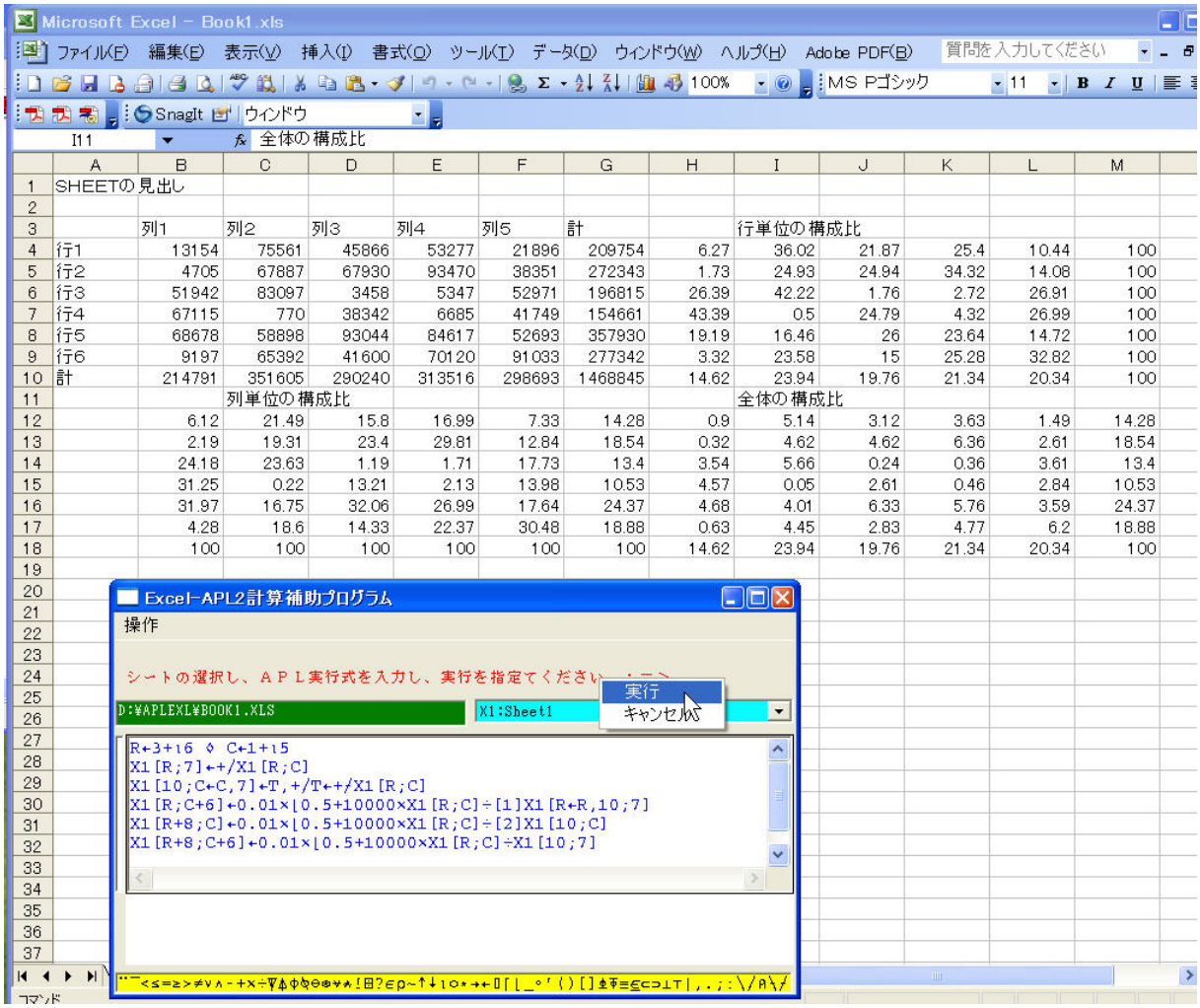
続けてプログラムの保存の確認が要求され、その後Ｅｘｃｅｌ画面表示が消去される。



メニューの ‘)ＯＵＴ’ 指定の場合は純ＡＰＬ環境とのデータ交換のため、計算結果を指定したディレクトリーにＡＰＬ転送データ形式ファイルで保存します。一連の作業はここで完了し、別の作業を入力ファイルや登録プログラムを指定して、操作を継続することができます。終了の場合は操作メニューの操作終了を選択します。

C）さまざまな計算

C－１．集計と一般的計算



ＡＰＬ式の解説：

```
R←3+⍳6 ◊ C←1+⍳5
X1[R;7]←+/X1[R;C]
X1[10;C←C,7]←T,+/T←+/X1[R;C]
X1[R;C+6]←0.01×⌊0.5+10000×X1[R;C]÷[1]X1[R←R,10;7]
X1[R+8;C]←0.01×⌊0.5+10000×X1[R;C]÷[2]X1[10;C]
X1[R+8;C+6]←0.01×⌊0.5+10000×X1[R;C]÷X1[10;7]
```

１行目：

⍳ｎ　は、１からｎ迄の整数の数列

３＋⍳６　は、４　５　６　７　８　９

Ａ←Ｂ　は、ＢをＡに代入すること。したがって　ＲとＣはそれぞれ計算対象の行と列のインデックス。

２行目、３行目：

Ｘ１　　は、ＳＨＥＥＴ１を二次元配列とみなしたもの。

＋／　　は、原始関数＋を原子作用子／で修飾した派生関数で、横軸(＝最後の軸)の合計

Ｘ１［Ｒ；２］＋Ｘ１［Ｒ；３］＋Ｘ１［Ｒ；４］＋Ｘ１［Ｒ；５］＋Ｘ１［Ｒ；６］

／とーの合成文字　は、縦軸(＝最初の軸)操作

応用：　体積←×／高さ　幅　奥行き

，(Ｄｙａｄｉｃ＝二項関数)　は、左引数に右引数を接続する原始関数。

Ｃ←Ｃ，７　で　Ｃ　は　２　３　４　５　６　７　になる。

４，５，６行目：

÷［ｎ］　　は、配列の各列または行を，第ｎ軸に沿って割り算（÷）する。

⌊　　は、原始関数ＴＲＵＮＣＡＴＥで、右引数の小数以下を切り捨てる。⌈　は切り上げ。

割り算の結果を小数２桁目で四捨五入して、小数以下２桁のパーセントにしている。

補足１：ＲやＣのインデックス値に条件式などの結果を用いると条件集計になる。

```
      □←R←T/⍳⍴T←⍉S←0=↑⍤¨0⍴⍤¨X1
4 5 6 7 8 9
      S
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0 0
0 1 1 1 1 1 0 0 0
0 1 1 1 1 1 0 0 0
0 1 1 1 1 1 0 0 0
0 1 1 1 1 1 0 0 0
0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
      T
0 0 0 1 1 1 1 1 1 0 0
      ⍳⍴T
1 2 3 4 5 6 7 8 9 10 11
      □←C←T/⍳⍴T←⍉≠S
2 3 4 5 6
```

⍴　　は、原始関数ＳＨＡＰＥでＭｏｎａｄｉｃなら右引数の配列（またはベクトルなど）の各軸の長さを求める。Ｄｙａｄｉｃなら右引数に左引数で指定したＳＨＡＰＥを与える。

０ρ¨Ｘ１　　は、ＳＨＥＥＴ１の各セルのＳＨＡＰＥを０（＝無、ｎｕｌｌ）にする。

↑０ρ¨Ｘ１　は、無にしたデータのＤＯＭＡＩＮ（＝領域、数値か文字か）を調べる。０なら数値領域、ブランクなら文字領域。すなわち、数値セルか、文字セルかを調べている。

∨／　　は、原始関数ＯＲ（∨）を‘関数／’の一般形式に従って、作用子／で修飾した派生関数。横軸に沿ってＯＲ条件を調べる。縦軸に沿う場合は／とーの合成文字を使用する。

／　　左引数が関数でなければ原始関数ＣＯＭＰＲＥＳＳＩＯＮ（＝圧縮、または抜き取り）で、右引数の配列（またはベクトル）を左引数で指定したＢｉｎａｒｙ（＝１と０で構成された）ベクトルの１に対応するものを抜き取る。

例：
```
      0  0  0  1  1  0  1／1  2  3  4  5  6  7
  4  5  7
```

補足２：規則的なインデック値を生成する場合は計算式を用いることもある。

例：　２×⍳５
```
  2  4  6  8  10
```

```
    ¯1＋2×⍳5
  1  3  5  7  9
```

註：マイナスの数値はＯｖｅｒ　Ｂａｒ　¯　を用い、原始関数ＳＵＢＴＲＡＣＴ　－　と区別する。絶対値は原始関数　｜を用いる。

例：　｜１　２　¯１　１５
```
  1  2  1  15
```

補足３：異なるＳＨＥＥＴ間での操作はＳＨＥＥＴ番号を指定すればよい。行列のサイズは一致する必要があるが、位置は一致しなくて用。

例：　Ｘ１［Ｒ；Ｃ］←Ｘ２［Ｒ＋５；Ｃ－１］＋Ｘ３［Ｒ－２；Ｃ＋３］

複雑な計算処理をする場合は異なるＳＨＥＥＴからのデータをあらかじめ３次元の変数（例：Ｓ）に移してから処理してもよい。どのように複雑な計算式も成り立つ。

例：　Ｓ←⊃Ｘ１［Ｒ；Ｃ］　Ｘ２［Ｒ＋５；Ｃ－１］　Ｘ３［Ｒ－２；Ｃ＋３］
```
      X1［R＋8；C］←×／［1］S
```

註：Ｓは３×（ρＲ）×（ρＣ）の３次元配列になる。×／［１］は第１軸（＝奥行き）に沿って積算すること。配列をファイルから読み込む場合も同様の処理をする。

補足４：その他の計算機能の例。

１．　べき乗　＊ｎ　　例：　１　２　３＊２　　は　１　４　９
　　例：　＊／２　３　２　　は　２＊３＊２　すなわち　５１２
２．　ルート　＊小数　例：　１　４　９＊０．５　は　１　２　３
　　例：　（＋／Ｕ÷ρＵ←Ｖ＊２）＊０．５　＜＝数列Ｖの標準偏差
３．　最大値　⌈／Ａ　は、Ａがベクトルならその最大値、二次元配列なら行毎の最大値
　　最小値　⌊／Ａ　は、Ａがベクトルならその最小値、二次元配列なら行毎の最小値
４．　剰余　　Ａ｜Ｂ　例：　５｜２３　¯９　１０．４５
　　３　１　０．５
５．　方向　　×（Ｍｏｎａｄｉｃ）　例　（×Ｖ）×└０．５＋｜Ｖ
　註：負を含む数列（または配列）の四捨五入。方向は　¯１（負）、０（０）、１（正）
６．　切捨て　└Ｖ　　例：　数列Ｖの最大公約数を求める。
　　切上げ　⌈Ｖ　　¯１↑Ｔ／⍳ρ∧／［１］Ｕ＝└Ｕ←Ｖ∘．÷⍳⌈⌈／Ｖ
　　註：／［１］は／－合成文字と同義。Ｖのそれぞれをの最大値の切り上げを承元とする整数数列で割り（∘．÷）Ｕに代入し、Ｕがすべて整数になる列の後ろから１つとる（¯１↑）。

```
      V←?6⍴10
      V
8 10 4 3 10 8
      V∘.÷⍳⌈⌈/V
 8 4   2.666666667 2    1.6 1.333333333  1.142857143  1     0.8888888889 0.8
10 5   3.333333333 2.5  2   1.666666667  1.428571429  1.25  1.111111111  1
 4 2   1.333333333 1    0.8 0.6666666667 0.5714285714 0.5   0.4444444444 0.4
 3 1.5 1           0.75 0.6 0.5          0.4285714286 0.375 0.3333333333 0.3
10 5   3.333333333 2.5  2   1.666666667  1.428571429  1.25  1.111111111  1
 8 4   2.666666667 2    1.6 1.333333333  1.142857143  1     0.8888888889 0.8
      U=⌊U←V∘.÷⍳⌈⌈/V
1 1 0 1 0 0 0 1 0 0
1 1 0 0 1 0 0 0 0 1
1 1 0 1 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0 0
1 1 0 0 1 0 0 0 0 1
1 1 0 1 0 0 0 1 0 0
      ∧⌿U=⌊U←V∘.÷⍳⌈⌈/V
1 0 0 0 0 0 0 0 0 0
      ¯1↑T/⍳⍴T←∧⌿U=⌊U←V∘.÷⍳⌈⌈/V
1
```

説明：？は乱数を発生させる原始関数。値が巨大でなければメモリーと処理時間はあまり問題ないが、メモリー不足でエラーになる場合は次のような式でも同じ結果が得られる。

```
I←⌈/V ◊ S←⍳0
T←⍎¨Iρ⊂'S←S,(T≠0)/T←I×∧/U=⌊U←V÷I ◊ I←I-1'
↑S
```

ＩをＶの最大値にセットして、Ｓを初期化する。
ＶをＩで割った値がすべて整数になるＩの値をＳに保存し、Ｉを１減らす式をＩ回繰り返し実行する。

Ｓの最初の値が最大公約数。
この方式では、最初に条件に合致する答えが発見されたとき、プログラムを中止できない。しかしコンピューターが速いので、答えは瞬間に求められる。これはＡＰＬＥＸＬがＦｕｌｌ機能のＡＰＬ実行環境でないからです。

Ｃ－２．時間、カレンダー、年齢などに関する計算

Ｅｘｃｅｌでは日付、時間に関するデータは１９００年１月１日を起点として１日を１として記録します。したがって整数部分が日付で、小数部分が時間です。またＥｘｃｅｌは時間を秒まで表現しますから、１日８６，４００秒で時間のデータを日、時、分、秒で捕らえるなら、次のような式を用いることができます。

```
      T⊤(×/1↓T←0 24 60 60)×1.5 0.78 1234.5678
 1   0 1234
12  18   13
 0  43   37
 0  12   37.92
```

すなわち、１．５は１日と１２時間、０．７８は１８時間４３分１２秒、１２３４．５６７８は１２３４日１３時間３７分３７．９２秒になります。日にちと時間を整数、小数に分けて別々に計算するには、

```
      0 1⊤1.5 0.78 1234.5678
1    0    1234
0.5 0.78    0.5678
```

とします。これはＡＰＬでの慣用的（Ｉｄｉｏｍａｔｉｃ）な使い方です。

Ｅｘｃｅｌに日にちや時間を入力するときには書式の形式でそのまま入力することもできますが、内部データ形式に変換して扱うときは、上の逆の操作をします。原始関数ＥＮＣＯＤＥ（⊤）の逆はＤＥＣＯＤＥ（⊥）です。

```
      (÷×/T)×(T←24 60 60)⊥⍞←⊃[1](12 0 0)(18 43 12)(13 37 37.92)
12 18 13
 0 43 37
 0 12 37.92
0.5 0.78 0.5678
```

註：⍞←はＡＰＬＥＸＬ上では使用できませんが、計算の途中の結果をＡＰＬ（製品版）の画面上に表示するときに用います。

ＡＰＬでは、システム変数“⎕ＴＳ”が現時間を年、月、日、時間、分、秒、ミリ秒の７個の数値からなるベクトルで表します。時間は規則的な数式で変換できますが、日付の変換はカレンダー使用する必要があります。１９００年から２００年間のカレンダーの定義は：

```
      U←(∧⌿S[⍳2;]=[1]1 0)∨∧⌿1 0↓S←T=⌊T←0.25 0.01 0.0025∘.×1899+⍳200
      ΔC←31,⊃(28+U),¨⊂10⍴5⍴31 30
      ΔC[⍳10;]
31 28 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 29 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 29 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
```

すなわち、グレゴリオ暦算出の数式を用いればいいわけです。しかしＥｘｃｅｌは１９００年をうるう年と間違えています。（註：Ｔ＝└Ｔは小数を切捨てて等しい（割切れる）の判定）

したがって、１日の誤差が出ないように、Ｅｘｃｅｌが用いていると思われる次の計算式を使用します。

```
      U←T=⌊T←0.25×1899+⍳200
      ΔC←31,⊃(28+U),¨⊂10⍴5⍴31 30
      ΔC[⍳10;]
31 29 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 29 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
31 29 31 30 31 30 31 31 30 31 30 31
31 28 31 30 31 30 31 31 30 31 30 31
```

すなわち、△Ｃは１９００年を起点とした２００年間の各月の日数で構成された２００×１２の二次元数値配列です。５⍴３１　３０は３１　３０　３１　３０　３１です。

曜日に関してもＡＰＬにはＥｘｃｅｌのような出来合いの仕組みはありません。２００６年３月１３日（月）はＥｘｃｅｌで３８７８９という値になります。したがって、月曜日を１とすれば、次に図解するように日付の値、１（＝１９００年１月１日）を７（＝日曜日）とすれば、カレンダーに関する問題はすべて解決します。

```
      1⌽⍳7
2 3 4 5 6 7 1
      10↑T←38789⍴⌽1⌽⍳7
1 7 6 5 4 3 2 1 7 6
      ¯10↑T
2 1 7 6 5 4 3 2 1 7
```

○と｜の合成文字はＤｙａｄｉｃ（＝２項形式）なら右引数を、左引数で指定した数だけ左向き（正の場合）または右向き（負の場合）に回転（ＲＯＴＡＴＥ）させます。Ｍｏｎａｄｉｃ（＝１項形式）なら左右の逆転（ＲＥＶＥＲＳＥ）になります。○と―は縦軸に沿っての回転と逆転になります。配列を扱う他の多くの関数と同様に［ｎ］の添え字を用いれば、３軸以

上の配列の任意の軸に沿った操作ができます。３８７８９は２００６年３月１３日に対するＥｘｃｅｌ上の値です。したがって、Ｔはその日から１９００年１月１日まで遡って、１を月曜日とする曜日の数列です。

ＡＰＬＥＸＬでは日付に関する処理を便利にするために、次のような手続きで、１９００年１月１日から始まる、年、月、日、曜日番号からなる４行７３０５０列の配列を作り△Ｄと定義し、いつでも使用できる状態にしておきます。

```
      ρY←(+/ΔC)/1899+ι200
73050
      ρM←(,ΔC)/,200 12ρι12
73050
      ρD←∈ι¨,ΔC
73050
      ρW←(+/,ΔC)ρ¯1φι7
73050
      ΔD←⊃Y M D W
      ¯14↑[2]ΔD
2099 2099 2099 2099 2099 2099 2099 2099 2099 2099 2099 2099 2099 2099
  12   12   12   12   12   12   12   12   12   12   12   12   12   12
  18   19   20   21   22   23   24   25   26   27   28   29   30   31
   5    6    7    1    2    3    4    5    6    7    1    2    3    4
      14↑[2]ΔD
1900 1900 1900 1900 1900 1900 1900 1900 1900 1900 1900 1900 1900 1900
   1    1    1    1    1    1    1    1    1    1    1    1    1    1
   1    2    3    4    5    6    7    8    9   10   11   12   13   14
   7    1    2    3    4    5    6    7    1    2    3    4    5    6
```

／　は、左引数が関数なら、その関数を修飾する作用子です。左引数が数値の場合、０と１からなるＢｉｎａｒｙベクターであれば抜き取り関数、２以上の正の整数であれば繰り返し（＝ＲＥＰＲＯＣＡＴＩＯＮ）関数になります。したがって、＋／△Ｃは、△Ｃを横軸に沿って（＝年毎に）日数を集計することですから、Ｙは各年の日数分年の値が連続するベクトルになります。Ｍは、２００年間の１月から１２月まで並んだ数列の要素を（＝月）を各月の日数分繰り返し、増幅し、ベクトルにしたものです。Ｄは各月のそれぞれの日数を上限とした１からの整数ベクトルを生成し、２００年分のならびにします。コはＹ　Ｍ　Ｄ　Ｗを配列化します。さらに祭日や、休日に関する情報を組み込めば、日付に関する処理はほとんど間に合いますが、これは必要に従って別途設定すればよろしいでしょう。

例１：　２００６年３月１４日から２００７年４月８日まで、土日を除いて（指定した日付も含めて）何日か。

```
      S←1++/(ΔD[4;]∈ι5)∧1=+\(⊂[1]ΔD[ι3;])∈(2006 2 14)(2007 4 8)
      S
300
```

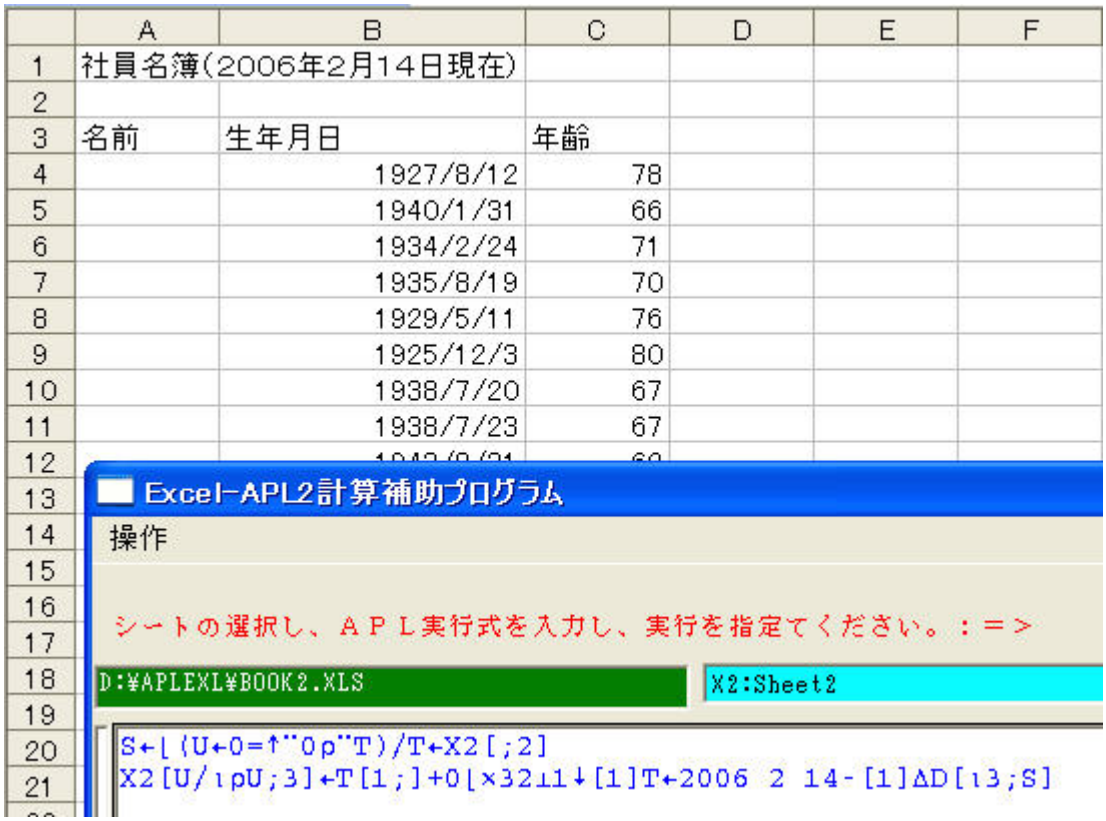
\　は、左引数が関数なら作用子ＳＣＡＮで、＋\　は派生関数累積で、右引数のベクトル（または配列の各ポジションの値が左端からそこまでの合計。たとえば、上のサンプルの場合日付が一致する場所が２箇所あり、ベクトルで示すことができる。

```
＋\0  0  0  1  0  0  0  0  0  1  0  0  0
   0  0  0  1  1  1  1  1  1  2  2  2  2
```

したがって＝１の部分は条件にあった場所に挟まれた範囲より１ポジション短い位置を示す。

ε　は、原始関数ＭＥＭＢＥＲで、左引数が右引数のどれかと等しい場所を示すベクトルを作る。

例２：　ＳＨＥＥＴ２の第２列に記録された名簿の生年月日から２００６年２月１４日現在の年齢を算出し、右隣の列に返す。



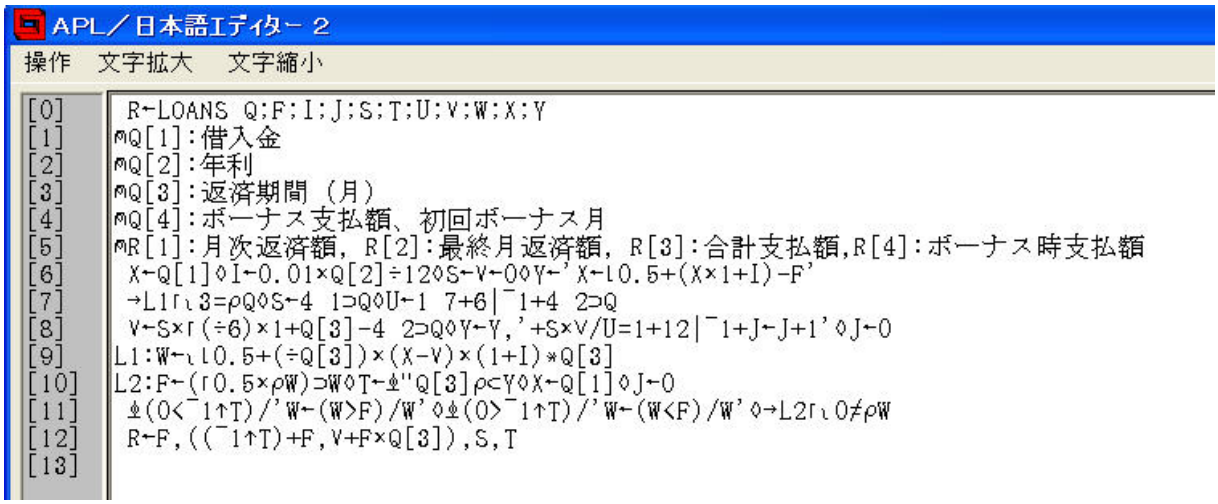
ＳＨＥＥＴ２の第２列の数値セルを取り出し（⌊で、念のために整数化した後）変数Ｓにセットする。同時に数値行の場所をＢｉｎａｒｙベクトルとして変数Ｕに記録する。△ＤのＳ列の年、月、日を　２００６年２月１４日から差し引き、変数Ｔに保存し、Ｔの１行目（年）を落とし、月日の差をそれぞれ３２進数として１０進数に変換し、正、０、負を判定し（＝Ｍｏｎａｄｉｃ　×）、０を上限としＴの１行目（＝年差）に加算し、ＳＨＥＥＴ２の第３列に返す。

月差がマイナスか、月差が０で日差がマイナスなら、３２進数（３１より大きい数であればなんでもよい）で変換すると、必ず負になる。その場合は年差から１を引かなければなりません。

註：ＳＨＥＥＴ２のインデックス値Ｓを算出するに当たり、ＡＰＬが内部的に保持する数値形式（＝浮動小数値）が確実に整数として取り扱われるように、普通は必要でない切り捨て（⌊）操作を行った。

Ｃ－３．金利に関する計算

ＡＰＬにおいては金利の計算は適用業務プログラムの範疇で、言語そのものにはこのような機能はありません。したがって、必要に応じて下に示すように、目的に適った関数をあらかじめ準備しておきます。ユーザー関数の定義はＡＰＬ２学習システム上でも行うことができます。



関数の定義に困難を感じる場合は、仲間に相談したり、出来合いのものをコピーさせてもらったり、調達することはさほど難しいことではありません。

使用例１：　年利＝３．２％
　　返済期間＝３０年
　　借入金＝３千万円

```
      3↑T←LOANS 30000000 3.2 360
129740 129752 46706412
```

すなわち、毎月支払額＝１２９，７４０円
　　最終月支払額＝１２９，７５２円
　　総支払額＝４６，７０６，４１２円

使用例２：　上記同一条件で、支払い開始２ヶ月目からボーナス時￥３００，０００支払いの場合。

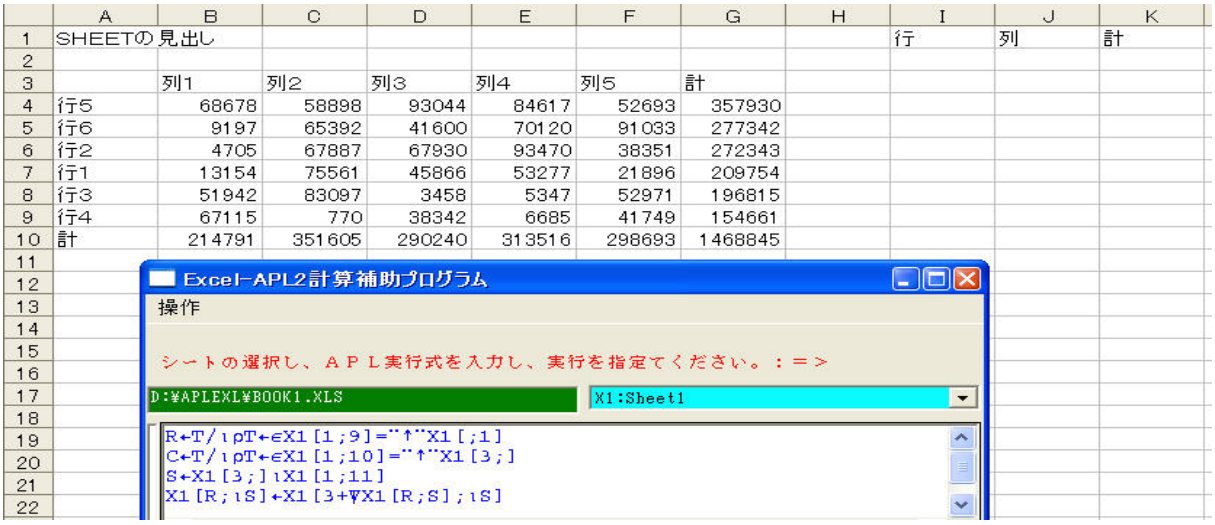
```
      4↑T←LOANS 30000000 3.2 360 (300000 5)
79940 79883 46778343 300000
```

すなわち、毎月支払額＝７９，９４０円
　　最終月支払額＝７９，８８３円
　　総支払額＝４６，７７８，３４３円
　　ボーナス時支払額：３００，０００円

Ｃ－４．漢字の処理とＣＳＶファイルの入出力について

大型コンピュータの場合を除いて、ＡＰＬでは漢字サポートは不十分です。弊社エーピーエル・コンサルタンツ作成のＡＰＬユーティリティー関数では独自に開発した方式で漢字をサポートしています。この方式はＷｉｎｄｏｗｓでの一般の方式との互換性がありますが、ＡＰＬＥＸＬにおいては、プログラムをできるだけ簡潔にするために、ＥＸＣＥＬの操作を利用したほうが合理的であると判断したところでは、あえてＡＰＬ側にその機能を設定してありません。したがって、たとえば漢字テキストを利用して、条件を判定するときなどには、Ｅｘｃｅｌの使用されていないセルや、すでにセル上に存在するテキストなどを参照して、利用する仕組みを採用しています。

例：Ｅｘｃｅｌ画面上の文字列を利用して条件に合致したセルを見つける。



ＡＰＬ式説明：

```
      □←R←T/⍳⍴T←□←∈□←X1[1;9]="□←↑"X1[;1]
ω   ωωωωωωω
  0   0   0   1   1   1   1   1   1   0
0 0 0 1 1 1 1 1 1 0
4 5 6 7 8 9
```

ＳＨＥＥＴ１の１列目の各セルの最初の１文字（１↑）が‘行’（Ｘ１［１；９］）であるか判定し、ネスティングされた深さ２の配列ベクトルを、関数ＥＮＬＩＳＴ（∈）で深さ１の単純ベクトルＴとする。それの長さを上限とする整数ベクトルを作りＴが１の場所に対応する数値を抜き出しＲとする。Ｒはみだし見出しが‘行’で始まる行のインデックスである。Ｃも同様。

```
      □←S←X1[3;]⍳X1[1;11]
7
```

ＳＨＥＥＴ１の３列目で漢字‘計’が最初に現れる列のインデックスをＳとする。

```
      3+□←⍒X1[R;S]
5 6 2 1 3 4
8 9 5 4 6 7
      ⍳S
1 2 3 4 5 6 7
```

合計列の降順インデックスで行を並べ替える。

ＡＰＬでＥｘｃｅｌの漢字を利用するときに、通常は問題になりませんが、内部で使用されているコードは、一般に使用されているＳｉｆｔ　ＪＩＳコードではなく、Ｕｎｉｃｏｄｅであることを理解しておかなければならない場合があります。たとえばＳＨＥＥＴ１の３行目の第２セルから第７セルが：

列１　列２　列３　列４　列５　計

として、ＡＰＬで読み込んで、コードを調べると確かにＵｎｉｃｏｄｅです。

```
      ⍴S←X1[3;1+⍳6]
6
      ⎕AF¨ S
 21015 65297  21015 65298  21015 65299  21015 65300  21015 65301  35336
      (4⍴16)⊤⎕AF∊ S
5 15 5 15 5 15 5 15 5 15  8
2 15 2 15 2 15 2 15 2 15 10
1  1 1  1 1  1 1  1 1  1  0
7  1 7  2 7  3 7  4 7  5  8
```

すなわち､コードの値が２５５を超えています｡すなわち１文字が１６ビットになっています｡これをファイルなどに書き出してＡＰＬで利用する場合はＡＰＬシステム関数ＣＴＫでＳｈｉｆｔ　ＪＩＳ　コードに変換する必要があります。逆の変換はＫＴＣを使用します。

```
      ⍴T←ΔCTK∊ S
22
      ⎕AF T
151 241 130 80 151 241 130 81 151 241 130 82 151 241 130 83 151 241 130 84 140 118
      16 16⊤⎕AF T
9 15 8 5 9 15 8 5 9 15 8 5 9 15 8 5 9 15 8 5  8 7
7  1 2 0 7  1 2 1 7  1 2 2 7  1 2 3 7  1 2 4 12 6
```

例：ＥｘｃｅｌのＳＨＥＥＴ（またはその部分）をＣＳＶファイルとして書き出す。

```
      S←2↓∊(⊂1↓⎕TC),¨1↓¨∊¨⊂[2]',',¨ΔCTK¨⍕¨X1
      T←S ΔFL 'D:\TEMP\TEMP.CSV'
```

ＡＰＬ式の説明：

```
      ⍕¨X1
ωωωωωωωωωω                                              ω         ω      ω

          ωω      ωω      ωω      ωω      ωω      ω               ωωωωωωω
ωω        13154   75561   45866   53277   21896   209754  6.27    36.02    21.87 25.4
ωω        4705    67887   67930   93470   38351   272343  1.73    24.93    24.94 34.32
ωω        51942   83097   3458    5347    52971   196815  26.39   42.22    1.76  2.72
ωω        67115   770     38342   6685    41749   154661  43.39   0.5      24.79 4.32
ωω        68678   58898   93044   84617   52693   357930  19.19   16.46    26    23.64
ωω        9197    65392   41600   70120   91033   277342  3.32    23.58    15    25.28
ω         214791  351605  290240  313516  298693  1468845 14.62   23.94    19.76 21.34
```

数値をすべて文字化します。漢字だけはＵｎｉｃｏｄｅ。他の文字はＡＳＣＩＩ。

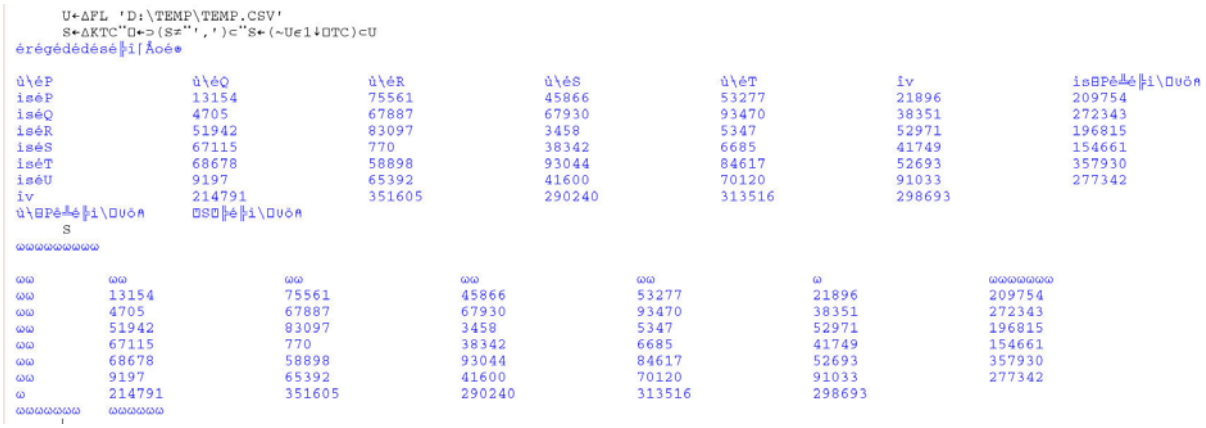
```
      ΔCTK¨⍕¨X1
érégédédésé╠î⌈Åoé⊛                                                  is             ù⍀    îv

                     ù⍀éP    ù⍀éQ    ù⍀éR    ù⍀éS    ù⍀éT    îv              is⊟Pê╧é╠î\⎕υöA
iséP                 13154   75561   45866   53277   21896   209754  6.27    36.02          21.87 25.4
iséQ                 4705    67887   67930   93470   38351   272343  1.73    24.93          24.94 34.32
iséR                 51942   83097   3458    5347    52971   196815  26.39   42.22          1.76  2.72
iséS                 67115   770     38342   6685    41749   154661  43.39   0.5            24.79 4.32
iséT                 68678   58898   93044   84617   52693   357930  19.19   16.46          26    23.64
iséU                 9197    65392   41600   70120   91033   277342  3.32    23.58          15    25.28
îv                   214791  351605  290240  313516  298693  1468845 14.62   23.94          19.76 21.34
```

Ｕｎｉｃｏｄｅ（漢字だけ対象）をＡＳＣＩＩ（Ｓｈｉｆｔ　ＪＩＳ）に変換する。

各セルの頭にコンマ文字‘,’を追加して、各行独立したレコードにする（ε ¨ ⊂［２］）。
各行の頭の‘,’文字１字ずつ捨てて、ＣＲ／ＬＦコード（１↓□ＴＣ）を追加した後、全体を１本の文字ベクトルにし、頭の２文字（余分なＣＲ／ＬＦ）を落とし変数Ｓにセットとする。Ｓを△ＦＬでファイルに書き出す。△ＣＴＫと△ＦＬは外部関数ＣＴＫとＦＩＬＥに名前を他と区別するために△文字をつけたもの。□ＴＣはシステム変数ＴＥＲＭＩＮＡＬ　ＣＯＮＴＲＯＬで、ＢＳ（バックスペース），ＣＲ（キャリッジ　リターン），ＬＦ（ライン　フィード）の３制御文字からなるベクトル。

このようにして作ったファイルはＥｘｃｅｌや他のシステムで、正規のＣＳＶファイルとして認識されます。ＣＳＶファイルをＡＰＬで読み込む場合は次のように行います。



式を拡大すると

```
U←ΔFL  'D:\TEMP\TEMP.CSV'
S←ΔKTC¨□←⊃(S≠¨',')⊂¨S←(~U∈1↓□TC)⊂U
```

で、書き出しと逆のプロセスになります。この式は一般のＣＳＶファイルをＡＰＬで読み込むときに汎用的に使用することができます。

ＡＰＬでは未知の対象に対して、ＡＰＬで解明できることが非常に多くあります。たとえば、ＡＰＬＥＸＬはＡＰＬで書きましたが、ＥｘｃｅｌのＸＬＳファイルからＳＨＥＥＴの枚数と固有の名前を見つけることができました。またＡＰＬＥＸＬのプログラムは画面設計の部分も含めて、ＡＰＬ式３００行弱に過ぎません。

ＡＰＬは難解な対象を知的なゲーム感覚で楽しみながら解き明かすことのできる、非常に珍しいプログラミング言語で、現代のさまざまなデータ処理の技術の基礎をなしています。ＡＰＬＥＸＬは単にＥｘｃｅｌの計算処理の代替案ではなく、できれば広く一般の方々にＡＰＬそのものに関心を持っていただくことを意図して作りました。

D）　その他のＡＰＬの主な機能

Ｄ－１．算術演算子の例（“ＡＰＬ２の世界”―著者編集（無償配布）より）

*R：　Exponential（指数関数）

Z←*R

ＺとＲは数値で、Ｚは自然対数 *e* のＲ乗の値。ただし *e* は約 2.7182818284590452 です。

```
      *0 1 2
1 2.718281828 7.389056099
```

⍟R：　Natural Logarithm（自然対数）

Z←⍟R

ＺとＲは数値でＲはゼロでないこと。Ｚは自然対数 *e* を底とするＲの対数。 ただし *e* は約 2.7182818284590452 です。

```
      ⍟1 2
0 0.6931471806
```

L⍟R：　Logarithm（一般対数）

Z←L⍟R

Ｌ、ＲおよびＺは数値でＬとＲはゼロでないこと。ＺはＬを底とするＲの対数。

```
      5 10⍟125 100
3 2
```

!R：　Factorial（階乗）

Z←!R

ＺとＲは数値でＲは正の整数。ＺはＲまでの全ての正の整数の積。

```
      !4 3 5
24 6 120
```

⌹R　　Matrix Inverse（逆行列）

Z←⌹R

Ｒは単純数値配列。Ｚは正則行列Ｒの逆行列。ＺはＲの行数が列数より大きい場合擬似逆行列(最小自乗法の意味での)。

```
      ⌹2 2⍴1 0 0 2
1 0
0 0.5
```

!R：　Factorial（階乗）

Z←!R

ＺとＲは数値でＲは正の整数。ＺはＲまでの全ての正の整数の積。

```
      !4 3 5
24 6 120
```

Ｄ－２．その他の論理、比較演算子

L=R：　Equal(等しい)
L≠R：　Not Equal(等しくない)

```
      Z←L=R
      Z←L≠R
```

ＬとＲはともに数値、またはともに文字で、Ｚはブール数。

```
      20 30 40 =40 30 20
0 1 0
      'ABC'='CBA'
0 1 0
      20 30 40 ≠40 30 20
1 0 1
      'ABC'≠'CBA'
1 0 1
```

L<R：　Less Than(小さい)
L≤R：　Less Than or Equal(小さいか等しい)
L>R：　Greater Than(大きい)
L≥R：　Greater Than or Equal(大きいか等しい)

```
      Z←L<R
      Z←L≤R
      Z←L>R
      Z←L≥R
```

Ｌ，Ｒは実数で、Ｚはブール数。

```
      20 30 40 <40 30 20
1 0 0
      20 30 40 ≤40 30 20
1 1 0
      20 30 40 >40 30 20
0 0 1
      20 30 40 ≥40 30 20
0 1 1
```

L∧R：　And(論理積)
L∨R：　Or(論理和)
L⍲R：　Nand(否定論理積)
L⍱R：　Nor(否定論理和)

```
      Z←L∧R
      Z←L∨R
      Z←L⍲R
      Z←L⍱R
```

Ｚ、Ｌ、Ｒはブール数で、Ｚは次のように要約できる。

論理積

∧	0	1
0	0	0
1	0	1

論理和

∨	0	1
0	0	1
1	1	1

否定論理積

⍲	0	1
0	1	1
1	1	0

否定論理和

⍱	0	1
0	1	0
1	0	0

~R：　Not(否定)

```
      Z←~R
```

ＺとＲはブール数で、ＺはＲが０なら１、１なら０。

```
      ~1 0
0 1
```

Ｄ－３．三角関数

```
∇
```

○R：　Pi Times(円周率倍)

```
Z←○R
```

ＺとＲは数値で、Ｚはπ とＲの積。

```
∇
```

LoR：　Circle Functions(円関数)

```
Z←LoR
```

Ｌ、ＲおよびＺは数値で、Ｌは下の表に示すように、三角関数、双曲線関数、ピタゴラス関数および複素数関数のどれをＲに適用するかを決める値で、Ｚはその結果値です。

L	関　　数	L	関　　数
		0	（1－R＊2）＊．5
¯1	Ａｒｃｓｉｎ　Ｒ	1	Ｓｉｎｅ　Ｒ
¯2	Ａｒｃｃｏｓ　Ｒ	2	Ｃｏｓｉｎｅ　Ｒ
¯3	Ａｒｃｔａｎ　Ｒ	3	Ｔａｎｇｅｎｔ　Ｒ
¯4	（¯1＋R＊2）＊．5	4	（1＋R＊2）＊．5
¯5	Ａｒｃｓｉｎｈ　Ｒ	5	Ｓｉｎｈ　Ｒ
¯6	Ａｒｃｃｏｓｈ　Ｒ	6	Ｃｏｓｈ　Ｒ
¯7	Ａｒｃｔａｎｈ　Ｒ	7	Ｔａｎｈ　Ｒ
¯8	－（8○R）	8	R≧0なら　－（¯1－R＊2）＊．5 R＜0なら　（¯1－R＊2）＊．5
¯9	R	9	Rの実数
¯10	＋R	10	Rの絶対値
¯11	0J1×R	11	Rの虚部
¯12	＊0J1×R	12	Rの位相

```
      1○1.5708
1
      2○.25
0.9689124217
```

例１：　半径１の円周の１度ごとのＸ，Ｙ座標地を求める。

```
      ρS←2 1°.○○(⍳360)÷180
2 360
      S[;⍳5]
0.9998476952  0.999390827  0.9986295348  0.9975640503  0.9961946981
0.01745240644 0.0348994967 0.05233595624 0.06975647374 0.08715574275
      ¯5↑[2]S
 0.9975640503   0.9986295348   0.999390827   0.9998476952  1
¯0.06975647374 ¯0.05233595624 ¯0.0348994967 ¯0.01745240644 0
```

Ｄ－４．作用子の役割

ＡＰＬは関数を動詞としたら助動詞に相当する原始作用子が６個あります。関数を修飾して派生関数を作ります。たとえば関数を　ｆ　とすると、

ｆ［１］　は、作用子Ａｘｉｓ（＝軸)。左右の引数を第１軸に沿って　ｆ　を実行します。

ｆ¨　　は、作用子Ｅａｃｈ（＝個別)。左右引数の配列の要素同士に対して　ｆ　を実行します。

ｆ／　　は、作用子Ｒｅｄｕｃｅ（＝低減）左引数を持たない（Ｍｏｎａｄｉｃ＝一項形式）は、右引数の配列の最後の軸に沿った要素をすべてに　ｆ　を作用させます。［ｎ］で軸を指定できます。　－が合成された文字は第１軸に沿っての操作になります。

左右の引数を持つ（Ｄｙａｄｉｃ＝２項形式）は、右引数の最後の軸に沿って左引数で指定したで指定した個数ずつの　ｆ　を実行します。［ｎ］で軸を指定できます。

例１：　３＋／１　２　３　４　５　６　７　８　９　１０
　　６　９　１２　１５　１８　２１　２４　２７
これを個数で割れば移動平均になります。

例２：　２－／右引数　は隣同士の差、

ｆ＼　　は、作用子Ｓｃａｎ（＝走査）Ｍｏｎａｄｉｃ（一項形式）しかありませんが、ｆ　の累積になります。［ｎ］で軸を指定できます。－が合成された文字は第１軸に沿っての操作になります。

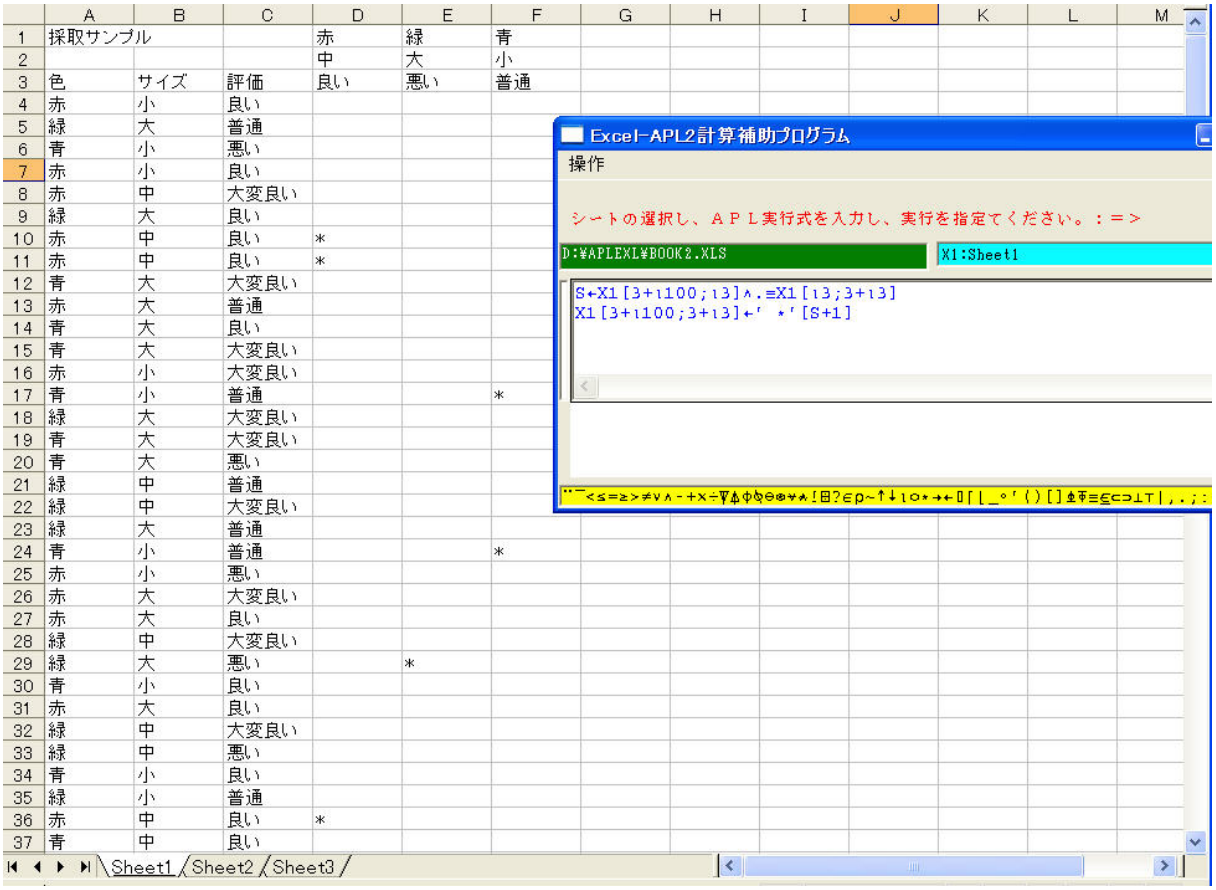
例１：

```
      □←S←?3 6⍴10000
5527 9658 2041 3472 8050 4742
5982 7079 6503 9064 9990 8469
5775 7706 8367 7837 3997 9383
      +\S
5527 15185 17226 20698 28748 33490
5982 13061 19564 28628 38618 47087
5775 13481 21848 29685 33682 43065
```

例２：　×＼１　２　３　４　５　６　７　８　９
　１　２　６　２４　１２０　７２０　５０４０　４０３２０　３６２８８
階乗の数列をつくります。次の操作と同じです。
　　！　１　２　３　４　５　６　７　８　９
　１　２　６　２４　１２０　７２０　５０４０　４０３２０　３６２８８

ｆ１．ｆ２　は、　作用子Ｏｕｔｅｒ　Ｐｒｏｄｕｃｔ（＝外積）。左引数の行と右引数の列のすべての組み合わせを　ｆ２　で処理し、その結果を　ｆ１　で処理します。

例１：複数の条件に合致するレコードをマークする：



ＡＰＬ式を拡大すると：

```
Ｓ←Ｘ１［３＋⍳１００；⍳３］∧.＝Ｘ１［⍳３；３＋⍳３］
Ｘ１［３＋⍳１００；３＋⍳３］←’＊’［Ｓ＋１］
```

例２：文字すべての組み合わせを接続させる

```
      (5 3⍴'ABCDEFGHIJKLMNO'),.,⊃[1]'123' '456'
ABC123   ABC456
DEF123   DEF456
GHI123   GHI456
JKL123   JKL456
MNO123   MNO456
```

補足：⊃［１］は右引数を１軸に沿って開く：　１　４
（この場合３行２列の配列を作る）　　２　５
３　６

⊃は右引数を最後の軸（＝横軸）に沿って開く：　１　２　３
（この場合２行３列の配列を作る）　　４　５　６

∘. ｆ　は、作用子Ｉｎｎｅｒ　Ｐｒｏｄｕｃｔ（＝内積)。左右引数のすべての組み合わせを　ｆ　で処理します。

例１：九九の表

```
      (⍳9)∘.×⍳9
1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

例２：　文字列の組み合わせを作る。

```
      ('ABC' 'DEF' 'GHI')∘.,'123' '456' '789'
ABC123 ABC456 ABC789
DEF123 DEF456 DEF789
GHI123 GHI456 GHI789
```