Tiny Basic for Windows 入門

tbasic.org [*1]

[2014 年 5 月版]

ここでは，Tiny Basic for Windows をインストールし，その操作法について慣れた後，実際にどのようにプログラムを作成していくか，簡単なプログラムを例にして説明をします。

目次

[*1] http://www.tbasic.org

1　プログラムとは

Tiny Basic for Windows （以下では tbasic と表します。）のプログラムの具体的作成を学ぶ前置きとして，プログラムについて簡単に説明をします。

1.1　プログラムとは

私達は，コンピューターを使うとき，「コンピューターで計算する」，「コンピューターで処理する」という言い方をよくします。しかし，正確には「コンピューターに計算させる」，「コンピューターに処理させる」と言った方が適切かもしれません。実際，例えば「ソロバンで計算する」と言った時は，自分で「実際にそろばん玉を手ではじいて計算をする」ことを意味しますが，「コンピューターで計算する」と言った時，実際の計算処理を私達が行っているわけではありません。コンピューターに「これこれの計算をせよ」という指示を出し，コンピューターはそれに従って，計算を行い，その結果を私達に返してくれているわけです。つまり 実は

コンピューターを使うとは，コンピューターに指示を出すこと

です。実際のコンピューターの使用では，多くの場合，一つ指示を出し，その結果を確認してから，次の指示を出すということで操作が進みます。つまり対話的に行われます。マウスのクリックも一つの指示ですし，キーボードでの入力も一つの指示と考えられます。その結果を確認しながら，次の操作に進みます。

しかし，ときには一括して指示を出し，連続的に処理を実行させる方が，都合が良い場合もあります。この処理を一括して行うための指示の集まりが**プログラム**です。マクロやスクリプトと言われるものもプログラムの仲間です。ですから

プログラムは
コンピューターに対する処理を一括して行うための
指示文書

と言えます。

プログラムは文書ですから，ある定まった意味を示す書式・形式に従って書きます。この形式のことを**言語**と言います。またその書式に従って書かれた文書を，その**言語のプログラム**と言います。C 言語で書かれているプログラムはCのプログラム，BASIC 言語で書かれているプログラムは BASIC のプログラムといった具合です。

コンピューターで使われている言語は色々あり，その分類の仕方も色々あります。まずプログラムの実行方法面からの分類として，コンパイラとインタプリタという分類があります。これについては，別項「コンパイラとインタプリタ」で書きましたのでそちらを参照してください。

さて次に，実際にプログラムを書く場合の心構えと言ったことを説明しましょう。実際の作成法や例については，項を改めて書きます。ここでは要点の概略を書きます。

1.2 プログラムの３要素

プログラムの持つべき重要な３要素を列挙してみましょう。それは

正しさ
速さ
分かりやすさ

です。このうち，「正しさ」と「速さ」については，言うまでもないことかもしれません。また「分かりやすさ」もそれに越したことはないでしょう。プログラムを書く場合これらがすべて揃えば大変良いプログラムと言えます。でも中々それは大変なことです。 ではプログラムを書く場合，どれに重点を置いて書くべきでしょうか。問題点を少し説明しましょう。

■ 正しいプログラム

正しいプログラムとは，目的とされる動作が正しく実行されるプログラムのことです。コンピューターのプログラムを書く場合，そのプログラムによって処理しようという目的があります。この処理は，普通その処理結果は分かっていないものです。実際，既に結果が分かっていれば，敢えて処理をする必要がありません。とすると，その結果が正しいと言うのは何で確信するのでしょうか。つまり，

プログラムが正しいことを検証する方法は何か？

ということです。コンピュータープログラムの場合，起き得るすべての場合をすべて試すと言った方法は現実的ではありません。実際，大抵の場合それは天文学的数を遥かに越えるのが普通だからです。ですから，

プログラムの正しさは，
論証で証明しなければならない

のです。この論証による証明は，当然ですが（自分を含め）他人を納得させるものでなくてはなりません。

■ 速く動作するプログラム

これがプログラムの重要な要件であることは，誰でも納得するでしょう。しかしいくつかの注意が必要です。最近のコンピューターは非常に高速ですから，どんなプログラムを書いても大体速く動作するような気がします。しかし色々な計算で現われる数は私達が想像するより，遥かに大きいものです。

例えば 30 人の人が一列に並ぶ方法が幾通りあるか考えてみましょう。これは 30! ですが，これは 33 桁の数です。大した数でないと思うかもしれませんが，33 桁の数だけ回数を繰り返す処理は，1 秒間に 1 兆回の処理ができる計算機で行っても 8 兆年以上かかることになります。

このように，ただ単にコンピューターを使うだけでは，高速に結果を出せる訳ではありません。ですから，計算しようとするものに対する詳しい分析を行い，よい計算方法（アルゴリズム）を使う必要があります。ですが，ここでも注意することがあります。良い計算方法とは良いアルゴリズムのことです。これはむしろプログラミング技法よりも数学的思考に属するものです。

細かいプログラミング技法による速度向上の努力も必要です。それはその言語の特徴や仕組みを知ることで実現されます。しかし，それよりも問題の数学的思考によるアルゴリズム改良の方が遥かに強力です。プログラミング技法による速度向上は，違う言語で同様な内容のプログラム書く場合，効果の無いこともあります。また，数年後のコンピューターの能力向上・高速化に伴って陳腐化することもしばしばあります。ですから，

動作の高速性は良いアルゴリズムを使用することで求める

べきでしょう。

■ 分かりやすいプログラム

実は，プログラムを書く上で最も注意を払うべき項目は，

分かりやすいプログラムを書く

ことです。分かりやすいプログラムとは，その目的，仕組みが秩序立てて整理され，自分が後で見たとき，すぐに分かるように書かれていることが第一です。しかしそれだけでなく，他の人が見たとき，容易に理解できるようになっていなくてはなりません。このようなプログラムはそれが正しく動くと言うことの確認になるだけでなく，後での，仕様変更，機能の改良，他の言語や他のコンピューター用に書き換えるときに，重要な要素になります。

このことを幾分標語的に表すと，

良いプログラマは正しいプログラムを書くだけでなく， それが正しく動作することを分かりやすく証明する必要がある

といえるかもしれません。この標語は私自身の努力目標でもあります。

このことを実現するためのプログラムの書き方が色々提唱されています。構造化プログラミングは定石とも言えるものです。この他，文芸的プログラミングと言った提唱もあります。書法や作法と言った本もあります。これらの詳細な内容はこのコースの内容を全く越えるものでし，私の手に負える範囲も越えています。で

すからここでは，取敢えず，「独り善がりにならない，分かりやすいプログラムを書くように努力しましょう」と言ったことと考えましょう。そのためには，他人のプログラムを読むと言うことも重要ですし，他人にプログラムを見てもらい議論するのも良いことです。

次に，プログラムを作る手順を説明します。

1.3 プログラムの作成手順

プログラムを作成して完成するまでの手順を簡単にまとめてみると次のようになります。

- 問題の把握
- アルゴリズムの決定
- コーディング（実際のプログラムを書く）
- テスト
- 完成

普通，プログラムの作成と言うと，この中のうちコーディング（Coding）を意味します。コーディングは元々符号化するという意味ですが，今の場合特定の言語用のプログラムを書くことを意味します。このためには，その言語の文法や性質，書き方を覚えなくてはなりません。このことはプログラミングの中で重要な要素ですが，しかしそれだけで良いプログラムが書けるわけではありません。

コーディング以後のことは，このコースの主要な話題ですので，ここでは，その前段階として，問題の把握とアルゴリズムの決定について，説明を加えましょう。

■ 問題の把握

プログラムを書くということは，当然何か目的があります。その目的を正確に把握し，何をしたいのかをはっきりすることが必要です。「簡単に出来る問題か，それともすごく難しい問題か」，その評価をする必要があります。

コンピューターは単純な仕事を高速に，正確に，何度も繰り返すような仕事が得意です。逆にそれ以外のものは不得意です。ですから，その問題を「非常に簡単な場合に限ったとき，それが手でも実際に可能で，その方法が分かっていて，しかも，本当に目的とすることはそれを何度も繰り返して，正確にやり続ければ完成する」ようなものなのか？

それとも，

「非常に簡単な場合に限ったときでも，その解決方法が分からないようなものなのか」

まず考えてみるべきです。

非常に簡単な場合を手計算で処理できないようなものは，恐らくコンピューターを使っても処理できないでしょう。このような場合，まず，簡単な場合を手で計算できるように，色々方法を考えてみる必要があります。

■　アルゴリズムの決定

アルゴリズムとは，物事を処理する具体的手順といえます。もう少し厳密に言うと，目的とする結果を，有限回の具体的手順で最終的に得ることができると保証された手順といえます。ここで具体的手順とは最終的にはコンピューター言語で表現できるような手順のことです。

コンピューターが理解できることは最終的にはすべてデジタルですし，その交信はデジタル信号です。我々がコンピューターに指示をする場合，勿論デジタル信号を送ると言ったこと意識をする必要はありませんが，最終的にはそれによる交信です。これはコンピューターへの指示がすべて，「**身振りや感情ではなく，文章化できる言葉によってである**」と考えても良いでしょう。

現在のコンピューター言語は色々な表現が可能ですが，しかしそれはあくまでも言葉で表現しなくてはなりません。しかし逆に明確の意味を持って表現されたものはかなりの場合，コンピューターで実現可能です。ですから，問題を把握し，簡単な場合手計算でできることが分かった場合，次に行うことは，

> 簡単な場合に手計算で行ったことを，他人に文章だけで説明する

ことを試みることでしょう。

この文章化の過程でその方法の正当性や有効性も説明することを考えましょう。そうすればもっと良い方法が思いつくかもしれません。

ここまでのことは極端に言えば，プログラム言語を全く知らなくてもできることですし，一般的な表現で言えば

> 物事を，正確に理解し，論理的に表現する

と言うことになります。このことはプログラミングとは関係なく一般的に重要なことで，国語教育，数学教育と言ったものの目的の一つでもあるでしょう。しかし，プログラミングを通じてこの能力を開発できるのも確かです。

■　コーディング

上のようにして，与えられた問題の解法が，「他人に明確に理解できるような手順により文章化された」とします。このときコーディングは，

> その文章をコンピューターの言葉で
> 書き直すことである

と言えます。日常的な言葉から，プログラム言語に変換する場合，勿論，プログラム言語の意味・約束事を学ぶ必要があります。そしてそれがこのコースのこれからの主要話題です。でもその前段階として「問題を他人に明確に理解できるような手順により文章化する」ことが必ず必要，もしくは最も重要であることを確認してください。

■ テスト

コーディングが終わると，次はプログラムの実行です。プログラムを実行して，「想定通り正しく動作するかどうか」を調べるのがテストです。

プログラムを実行したとき，何らかのエラーメッセージが出された場合は，プログラムは言語的等のエラーを含みます。この場合はまずエラーメッセージをよく確認して，エラーが出なくなるまで修正をします。この修正は根気のいる仕事ですが，重要な作業です。

このような作業を経て，エラーメッセージが無くプログラムが実行されたとします。しかしこれで終わりではありません。実はここからが本当の意味でのテストです。プログラムがエラーなく実行されたとしても，プログラムが想定通りの動作をしたことの保証にはなりません。1.2 で「プログラマは正しく動作することを証明する必要がある。」と書きましたが，実際のコードでこのことの確認を行う必要があります。この確認は次の方法で行うのが良いでしょう。

(1) 理論上の確認：
実際のコードが目的のアルゴリズムを実現するものであることを，プログラムの仕様と照らし合わせ論理的に確認します。

(2) 実験での確認
プログラムは実際の計算処理のために作成されます。ですから，そのいくつかの処理を実際に実行させて，想定される結果と一致するかをチェックします。すべての場合で一致すれば完全に確認されたことになりますが，実際にはそのようなことができないのが通常です。ですから，想定されるいくつか例に限ってチェックすることになります。そのチェックするデータのことをテストデータと言いますが，テストデータは次のようにして選びます。
・ 標準的に想定されるデータ
・ 境界値データ：仕様上は許されるが特殊な状況のデータ
これらのいくつかのデータを選び，プログラムでの結果と実際に別な方法で計算された結果と一致するかを確認します。

(1) で完全に確認されれば，(2) は不要なはずですが，再度の確認として (2) があります。　これが確認されればプログラムは完成です。このようにテストは

プログラミングの総仕上げ
最後の確認

です。おろそかにしないようにしましょう。

1.4 まとめ

最後のまとめをしましょう。

> プログラミングはあくまでも道具です。
> それを表現する対象に対する深い理解が無ければ，
> 良いプログラムを書くことは出来ません

勿論，プログラミングを学ぶことを通して，対象に対する理解が深まることも多くあります。家庭教師をして，自分自身理解が深まったと言う話は良く聞きます。コンピューターは真面目ですが，ある意味で物分りの悪い生徒です。そのため根気よく丁寧に指示をする必要があります。

プログラムを書き，実行する過程でさらに学ぶべきことを発見することは沢山あります。プログラムを書きながら，目的とする対象の理解とプログラミングの理解を螺旋的に深めていくのが良いでしょう。

2　プログラムの構成要素

ここでは，具体的なプログラムとはどのようなものなのかの概要を説明します。「プログラムとは」の項で

プログラムはコンピューターに対する処理を一括して行うための指示文書

ということを書きました。文書ですからある確定した意味を持つ書式・形式に従って書きます。この形式が言語でした。言語によってその書き方も違いますし，従って，プログラムのもつ形式も違います。このコースはBASIC 入門ですから，BASIC プログラムについて説明します。BASIC はプログラム言語の中で，手続き型言語と言われるものので，C，Pascal，Fortran などもこの仲間です。ですから，具体的表現は違いますが，基本的考え方は似ています。

2.1　プログラムの構成要素

プログラムは指示文書ですから，指示の集まりです。一つ一つの指示を文（statement）と言います。ですから，

プログラムは文の並び

といえます。文は作用の視点から分けると，実行文と非実行文に分類されます。

■　文の作用

- 実行文は，その文が実際の実行を指示するもの。単純実行文制御文
- 非実行文は，実行の指示ではなく，実行文によって利用されるもの。

また，プログラムを行の集まりと考えると， 単純文とブロック文に分類されます。

■　文と行

- 単純文：1 行で完結する文。
- ブロック文：何行にも渡る文。

という分類も出来ます。

文はいくつかの要素から構成されます。

■　文の構成要素

- キーワード：決まった意味をもつ文字列や記号,
- コマンド：指示を意味する特別なキーワード
- 式：目的によってプログラマーが色々構成する

具体例で説明した方が良いでしょう。

2.2 具体例

よく，13 日の金曜日は不吉と言われます。このことを気にしてみると，なんだか 13 日の金曜日は割合あるように感じられます。

実際にどのくらいの割合で，13 日の金曜日があるのか調べてみたくなりました。作業自体は簡単で，カレンダーを根気よく調べれば確認できます。しかし面倒です。このようなことこそ計算機が得意な分野です。そこで 13 日の曜日を数えるプログラムを作ってみました。それが以下のプログラムです。

tbasic の実際のプログラムでは行番号は使いませんが，以下では説明のため，行番号 2 桁と 2 個の空白を付け加えています。ですから左から 4 文字は実際のプログラムでは無いものです。

```
01  ' １３日の曜日を数える
02
03  TDay = 13    :' 日にちの指定 (1〜28)
04  Dim DayNum(7)
05  For i=2001 to 2400
06     For j=1 to 12
07        DateStr$ = Str$(i)+"/"+Str$(j)+"/"+Str$(TDay)
08        DayNumber = DayOfWeek(DateStr$)
09        DayNum(DayNumber)=DayNum(DayNumber)+1
10     Next j
11  Next i
12  Print TDay;"日の曜日の出現割合（%）"
13  Print "  日     月     火     水     木     金     土"
14  For i=1 to 7
15     Print using "##.##";DayNum(i)/48;
16     Print "  ";
17  Next i
18  Print
19  End
```

このプログラムの説明をしましょう。このプログラムの意味・内容の説明はプログラム例に譲ることにして，ここでは主に形式についての説明をします。個々の文の意味や使い方の説明はこれからですので，ここでは雰囲気を掴むことが目的です。

このプログラムは全部で 19 行のプログラムです。1 行目から順に説明しましょう。

```
01   ' １３日の曜日を数える
```

これは**コメント文**と言われるもので，'がコマンドです。'コマンドの意味は「以下を行末まで無視せよ」というものです。何も実行しない文です。ですから，コメント文は 1 行からなる**非実行文**といえます。

コメントはコンピューターが無視するということを利用して，我々に対する説明を書きます。ここでは，このプログラムの目的を書いています。

```
02
```

2 行目は実は何もありません。**空白行**です。この行は実際には何の作用もしません。ただ見やすさのためだけのものです。空白行はどこにいくつ入れても動作に何の関係もありません。見やすさのため適当に空白行を使うことができます。

```
03  TDay = 13    :' 日にちの指定 (1〜28)
```

この行には，実は 2 つの文が書かれています。コロン : は行区切りと同じ意味を持つキーワードです。ですからこの行は 1 行で 2 行分の内容になっています。前半の TDay = 13 は代入文と言われる文です。 = が実はコマンドで，左側の TDay は数値変数，右側の 13 は式ですが，実際は 13 という数値定数です。TDay = 13 は 13 と言う数値を TDay と言う数値変数に代入しています。これは実行文といえます。コロン : の右側は 1 行目と同じコマンド ' を使ったコメント文です。

昔の N88BASIC 等ではコロンを使って 1 行にいくつもの文を書くことがしばしば行われていました。これは，当時の N88BASIC が完全に 1 行単位で機械語への変換を行っていたため，2 行に書くより 1 行にまとめた方が実行速度が上がったことによります。今でもその名残が残っているプログラムを見ることがあります。

しかし現在の BASIC ではそのようなことは無いようです。少なくとも tbasic については，全くそのようなことはありません。コロンを多用して 1 行に多くの文を書いても，それを別の行に並べて書いても，実行速度には関係しません。

ですから，普通はコロンを使う必要はない訳です。現在コロンを使って， 1 行にいくつかの文を入れる理由は見やすさの問題だけです。上の場合もそれに当たります。

```
04  Dim DayNum(7)
```

この行は 宣言文といわれるもので，Dim がコマンドです。その後の，DayNum(7) は配列変数というものです。この Dim 文はこれから**配列変数** DayNum(1)〜DayNum(7) を使うと宣言しています。この文は宣言しているだけで，この文だけでは，実際の動作はしませんので，非実行文です。

```
05  For i=2001 to 2400
06     For j=1 to 12
07        DateStr$ = Str$(i)+"/"+Str$(j)+"/"+Str$(TDay)
08        DayNumber = DayOfWeek(DateStr$)
09        DayNum(DayNumber)=DayNum(DayNumber)+1
10     Next j
11  Next i
```

5 行目から 11 行目は 1 つのブロック文です。これは For Next 文といわれる文で，原則的に複数行で書くものです。コロンを使って 1 行に書くことも出来ますが，標準的ではありません。5 行の For から 11 行の Next i で文が完結します。 For 文は繰り返し構文と言われる，似た動作を何度も繰り返すとき使います。

5 行の i=2001 to 2400 は i を 2001 から 2400 まで次の行から 11 行目まで繰り返すという指示を意味します。

実際は 6 行目から 10 行目までが繰り返す文です。このように文の中に，また文が入ることのできる構文もあります。今の場合，6 行目から 10 行目がさらにまた For Next 文 になっています。このような場合複数の繰り返し文が「**入れ子になっている**」といった言い方もします。

5 行目と 6 行目が行の先頭文字の位置が違うこと気が付くと思います。6 行目と 7 行目も先頭文字の位置が違います。これらを**字下げ（インデント）**と言います。コマンド間や先頭に空白を入れても，実際の動作には関係しません。ですから上の文を

```
05  For i=2001 to 2400
06  For j=1 to 12
07  DateStr$ = Str$(i)+"/"+Str$(j)+"/"+Str$(TDay)
08  DayNumber = DayOfWeek(DateStr$)
09  DayNum(DayNumber)=DayNum(DayNumber)+1
10  Next j
11  Next i
```

と書いても動作は同じです。しかし，今の場合上のように字下げをして書くのが標準です。字下げはプログラムでの一つのブロックを明確するためにするものです。ブロック文の場合，ブロックの中身を字下げをすることで，ブロックの範囲が明確になります。５行も６行もブロック文 For Next 文の始まりですので，２段階の字下げが行われています。

字下げの場合何文字分の空白にするかは決まりはありません。私の場合は一応，３文字空白としています。

```
07          DateStr$ = Str$(i)+"/"+Str$(j)+"/"+Str$(TDay)
08          DayNumber = DayOfWeek(DateStr$)
09          DayNum(DayNumber)=DayNum(DayNumber)+1
```

7 行から 9 行が実際に繰り返す実行文です。7，8，9 行ともすべて代入文です。代入文は = の左辺が数値あるいは文字列変数，右辺が式の形をしています。式は色々なものから構成されるものですから，左辺が単純で右辺が複雑な形になるのが普通です。

7 行の = の右辺 Str$(i)+"/"+Str$(j)+"/"+Str$(TDay) が式ですが，この式は関数 Str$()，数値変数 i，演算子 +，文字列定数 "/"，数値変数 TDay によって構成されています。

8 行の = の右辺 DayOfWeek(DateStr$) は関数 DayOfWeek()，文字列変数 DateStr$ によって構成されています。

9 行の = の右辺 DayNum(DayNumber)+1 は配列変数 DayNum()，数値変数 DayNumber，演算子 + ，数値定数 1 によって構成されています。

```
12  Print TDay;"日の曜日の出現割合（%）"
13  Print "  日      月      火      水      木      金      土"
```

12，13 行は Print 文です。Print 文は Sample にある Direct.tbt を実行した人にはお馴染みですね。Print 文は色々な使い方がありますが，12 行は 数値変数 TDay と 文字列定数"**日の曜日の出現割合（%）**" を Print 文での区切り文字セミコロン ; を使って分離しています。

13 行は単純な Print 文です。文字列定数" 日　月　火　水　木　金　土" だけを使っています。

```
14  For i=1 to 7
15     Print using "##.##";DayNum(i)/48;
16     Print "  ";
17  Next i
```

14 行から 17 行は For Next 文です。For Next 文は頻出の構文です。ブロック文ですから 15，16 行が字下げされています。

15 行は Print 文ですが using というキーワード付です。これは書式制御指定というもので，続く文字列定数 "##.##" が書式指定文字です。その後，区切り文字 ; そして，式 DayNum(i)/48 が続きます。最後にセミコロン ; があることに注意して下さい。このような使い方もあります。 Print 文で最後が セミコロンだと画面表示後 改行をしません。

16 行も改行なしの Print 文です。

```
18  Print
```

18 行は今度は，Print のみですが，これは改行のみを指示しています。

```
19  End
```

これは End 文というものです。文字通り終わりと言う意味で，プログラムの終了を指示します。End 文は最後に一つだけ書くという規則を持つ BASIC もありますが，マイクロソフト系の BASIC はどこにいくつ書いても構いません。tbasic も End 文もどこにいくつ書いても構いません。

2.3 最後に

「習うより慣れろ」と言うことわざがあります。コンピューターの場合もその典型とも言えますが，プログラミングの場合は少し注意が必要です。プログラムは厳格な規則に従って書く必要がありますから，ある程度の基本的事項は確実にしておくべきです。「マニュアル百遍」という言葉もあります。

3　式とその構成要素

プログラムは文の並びでした。文は色々なものから構成されますが，その中で最も基本的で，またプログラムの中で最も多く使われるのが式です。ここではその式の作り方や基本的性質について説明します。ここでの説明は tbasic 向けの説明ですが，特に断らない限り，BASIC 一般に適用できます。

式はいくつかの基本的要素を組み合わせて作ります。その基本的要素とは

- 定数
- 変数
- 演算子
- 関数

です。

BASIC 言語の中で扱う対象は色々ありますが，キーボード，画面，ファイルなど出入力を通じて最終的に扱う対象は「数」と「文字列」です。変数や式などは BASIC プログラムの中で色々に表現されますが，それらは常に値もしくは内容といったものを持っていて，（いくつかの例外はありますが,）それはコンピューター内部では「数」または「文字列」のいずれかの形式で保存されています。

まず定数について説明しましょう。

3.1　定数

定数とは具体的に定まった対象のことです。

3 とか 1.5 とか −100 とかは定まった数です。このような具体的な数を数値定数と言います。これに対して，「BASIC」とか「Computer」とか「今日は晴れです。」といった，定まった文字や記号の列を文字列定数と言います。

文字列定数と言う言い方に違和感を感じるかも知れませんが，元々この「定数」は constant の訳語で「数」の意味はありません。文字列定数は英語で string constant と言います。また，数値定数は英語で numeric constant と言います。

言語によっては文字列と数値の区別が厳密でないものもありますが，BASIC を含む多くの言語は型の違った対象を厳格に区別します。ですから，数値定数と文字列定数の表し方には定まった規則があります。

数値定数を表すには，普通に数字をそのまま書きます。文字列定数の場合は少し注意が必要です。BASIC で文字列定数を表すときは，その文字や記号の列の前後に引用符"をつけてその間が文字列定数の中身であることを示します。ですから，「BASIC」と言う文字列を表すには，"BASIC" と書きます。同様に，「Computer」と言う文字列を表すには，"Computer" と書きます。同じように「今日は晴れです。」と言う文字列を表すには，"今日は晴れです。" と書きます。まとめると

- 数値定数は普通に数字を書く。
- 文字列定数は引用符 " で囲む。

となります。

例えば，「3」 と言う数を表すときは，普通に 3 と書き，「3」 という文字列を表すときは "3" と書きます。"3" のように 1 文字を列と言うのも少し変ですが，1 文字も文字列と言います。何もないものを表すときは "" と表し，空の文字列と言います。多少の例外はありますが，BASIC では数と文字列の型の違いには厳格です。

数値定数や文字列は定まったものですが，内容が変化する対象もあります。それが変数です。次に変数について説明しましょう。

3.2 変数

変数とは内容が変化し得る対象のことです。

「数」用の変数と，「文字列」用の変数と 2 種類あり，「数」用の変数には数，「文字列」用の変数には文字列だけしか入れられません。変数は定数を入れる「箱」とも考えられますが， 1 つの変数には 1 度には 1 つしか定数を入れられませんので，「座席」と考えることもできます。数値定数を入れる箱（または座る座席）を数値変数，文字列定数を入れる箱（または座る座席）を文字列変数と言います。

定数の場合と同様に，文字列変数と言う言葉に違和感がありますが，変数という言葉は英語の variable の訳語でもともと「数」の意味はありません。

変数は定数と違い，その値や内容で区別することは出来ませんから，名前を付けて区別します。電卓で値を記憶する メモリー１ とか メモリー２ とかあるのもありますが，これらは名前がメモリー 1，メモリー 2 という変数と考えられます。この場合は名前は既に決まっていて勝手に名前を付けて使うことは出来ません。これに対して，BASIC では変数は適当に名前を付けて使います。また予め名前のついた変数はありませんので，変数を使うには名前を付ける必要があります。

変数の名前の付け方にはいくつかの考えがありますが，tbasic では次の単純な規則で付けます。

- 数値変数の名前は英字で始まり英数字から作られた文字列です。(20 文字以内)
- 文字列変数の名前は英字で始まり英数字から作られた文字列の最後に$を付けたものです。
  (20 文字以内)

数値変数，文字列変数とも名前は適当に付けて良いのですが，変数名として使えないものもあります。それは予約語と言って，BASIC で既に使っているものです。例えば Print は画面表示用の BASIC のコマンドでしたが，これと同じ名前を変数名として使うことは出来ません。

どんな予約語があるかを覚えるのは中々大変ですから，予約語にならないと推測できる名前を付けると良いでしょう。それは

- 例に使われて良く見る名前，例えば `i`，`j`，`k` を使う。
- また予約語が英語であることを利用して，
  日本語のローマ字表記のもの，例えば，`Kekka`，`Kosuu`, などです。

また，英字は大文字小文字を使えますが，実行上は大文字・小文字の区別はありません。ですから ABC と abc は同じ変数を表します。

まず，数値変数の名前の例を挙げましょう。

例 3.1 (数値変数名)**.**

```
a
b
i
x
A1
Sum
Result
Kosuu1
Kekka1
```

これらは，数値変数名として使えるものですが，次は変数名として使えないものです。

例 3.2 (数値変数名として使えない例)**.**

```
1A     … 最初の文字が数字ではいけません
A,C    … 「,」は使えません
A+     … 「+」も使えません
あ 1    … 日本文字は使えません
B  X   … 空白「 」も使えません
```

次に文字列変数の例を挙げましょう。

例 3.3 (文字列変数)**.**

```
A$
B$
Moji$
Namae1$
St$
```

これらは文字列変数の名前として使えるものです。

これで変数の名前の付け方は大体理解できたと思います。

では変数を使うのはどうしたら良いのでしょうか？
それは名前を決めてプログラムの中で使えば良いのです！
それだけです。

BASIC は初めての変数名を見つけると，直ちにその名前を持った変数をコンピューターのメモリの中に用意します。 用意したとき，変数の箱の中身は数値変数は 0，文字列変数は空です。

以上のことをまとめると

- 数値定数を入れる箱（または座席）を数値変数
- 数値変数の名前は英字で始まり英数字から作られた文字列
- 数値変数の初期値は 0
- 文字列定数　を入れる箱（または座席）を　文字列変数
- 文字列変数の名前は英字で始まり英数字から作られた文字列の最後に$を付けたもの文字列変数の初期値は空となります。

式は定数と変数から始めて演算子や関数を使って構成します。次に演算子について説明しましょう。

3.3 演算子

演算子は式にある操作を施して新たな式を作るものです。式は必ず値を持っていて，その値に対して操作をします。式の持ち得る値は，数値，文字列，それに真理値です。

式は演算子，変数，定数，関数などをその規則に従って構成します。そして演算子や関数の引数として式を使う場合，この種類が一致していなくてはなりません。またこの構成で適当に括弧 (および括弧) を使って，演算の順序を分かりやすくすることが出来ます。

これから説明する個々の演算子については，ヘルプにある Reference Manual の演算子の項に更に詳しい説明があります。必要な場合はそちらも参考にして下さい。

対象とする式の持つ値に応じて演算子は分類することが出来ます。この分類に従って説明を進めましょう。

3.3.1 数値演算子

数値を値に持つ式を対象とする演算子として次のものがあります。普通の数の計算に使うものとほぼ同じものと，BASIC など計算機言語で特有なものとがあります。以下が数値演算子の一覧です。

項数	演算子	意味	使用例	説明
1	+	正符号	+A	A と同じ
1	-	負符号	-A	マイナス A
2	^	冪	3^4	3 の 4 乗
2	*	積	2 * a	2 掛ける a
2	/	除法	3 / 5	3 割る 5
2	+	和	Ab + 3	Ab 足す 3
2	-	差	Cd - F	Cd 引く F
2	¥	商	17 ¥ 5	17 割る 5 の商，今の場合 3
2	mod	剰余	17 mod 5	17 割る 5 の余り，今の場合 2

ここで，加減乗除のうち，×と÷はキーボードにないので， * と / を使います。指数を示す場合は ^ を使います。最後の２つ ¥と mod は見慣れないかもしれませんが，覚えておくと便利です。

上の表で，項数という用語が書かれていますが，項数は，演算子を使うとき，必要とされる対象の個数です。単項演算子（１項演算子）では，演算子を使うとき対象が１つです。例えば，

```
-A
```

のように，マイナス演算子 - を使うとき，マイナスの後に 1 つの対象 A があるだけです。同じ記号 - を

```
A-B
```

のように，２項演算子として扱うことも可能です。この場合，A と B と２つを対象にする演算子となります。また，演算子の間には適宜空白を置いても構いません。ですから，上の式は，それぞれ

```
- A
A - B
```

としても同じです。

実は，演算子を使う場合，厳密には，全体を括弧で囲む必要があります。ですから上の例では，

```
(-A)
(A-B)
```

と書くのが厳密な書き方です。また空白を入れて，

```
( - A)
( A - B )
```

等としても同じです。ただ，複雑な式でそこに含まれる括弧をすべて書いたり，空白を多く挿入すると却って見づらいこともあります。ですから，見易さのために適宜，括弧を省略します。また空白は見易さのために適宜挿入します。BASIC に限らず一般にプログラミング言語では括弧は (と対応する) と対にして，省略することができます。また演算子や括弧の間に空白を挿入しても同じです。

いくつか例を見てみましょう。

例 3.4 (数値式)**.**

```
1 + 2
A * B
-2^2
-(2^2)
(-2)^2
3 + A/B - C
(3 + A)/B - C
(((3 + A)/B) - C)
((3 + A)/(B - C))
(SUM1 + SUM2)/2
```

これらは，すべて正しい数値式です。括弧をすべて厳密に入れると見づらくなりますが，あまり省略すると，意味が異なってしまったり，誤解を招くこともあります。煩雑にならない範囲で，むしろ括弧を積極的に使うようにしましょう[*2]。

[*2] Excel を持っている人は，-2^2 を Excel で計算してみましょう。実は，結果は 4 になります。他方，tbasic で -2^2 を計算すると，-4 になります。ですから，このような式は，どちらで計算しても同じになるように，$-(2^2)$ または，$(-2)^2$ と書くべきでしょう。

次のものは正しい数値式ではありません。

`1 : 2` … **:は演算子としては使えません。**
`1 *+ A` … ***+ は演算の意味が判りません。**
`1 2 * 41` … **数値定数 12 で 1 と 2 の間に空白を入れてはいけません。**
`(AB +CD)/E)` … **括弧の使い方が正しくありません。括弧の省略は（と）の対で省略します。**

最後に，式の値という言葉を説明しましょう。式は色々なものから構成されますが，出発点は定数と変数です。定数は定まった数か文字列でしたし，変数もその値を持っています。ですからその値から演算の意味に従って，

> 演算を行うと最終的にはひとつの値が定まります。
> これを式の値または内容と言います。

例えば

`1 + 2`	の値は	3
`A * B + 2`	の値は，A の値が 3 で　B の値が 4 ならば，	14
	の値は，A の値が 5 で B の値が -2 ならば，	-8

となります。

3.3.2 文字列演算子

文字列に対する演算子は結合演算子 + のみで簡単です。

演算子	意味	使用例	説明
+	結合	"Ba"+"sic"	Ba と sic を結合して BASIC を作る

例を見てみましょう。

例 3.5 (文字列式)**.**

```
"Bas"+"ic"
A$ +"ic"
Ab$ +Cd$
```

これらはすべて正しい文字列式です。

これらの式についても値または内容があります。文字列変数を含む場合は，変数の内容に依存します。例えば上の例で，

"Bas"+"ic"	の値は，	BASIC　です。
`A$ +"ic"`	の値は，	A$ の内容が Graph ならば，Graphic　ですし，
	A$ の内容が Bas ならば，	BASIC　となります。

3.3.3 論理演算子

式の値として持つ３つめのものとして。真理値があります。真理値は真または偽をとるものです。真理値を扱う場合，実際には数値を使って表す処理系もありますが，tbasic では別に扱っています。しかしこの真偽を実際に代入して使うことはありませんから，使い方としてはどちらの方法でも同じです。

論理演算子としては次のものがあります。

演算子	意味	使用例	説明
<=	不等号	A <= B	A が B 以下なら真，そうでなければ偽
>=	不等号	A >= B	A が B 以上なら真，そうでなければ偽
<	不等号	A < B	A が B 未満なら真，そうでなければ偽
>	不等号	A > B	A が B を越えれば真，そうでなければ偽
<>	不等号	A <> B	A と B が異なれば真，そうでなければ偽
=	等号	A = B	A と B が同じなら真，そうでなければ偽
not	論理否定	not (A > B)	(A > B) でなければ真，そうでなければ偽
and	論理連言	(A=B) and (C=D)	(A=B) かつ (C=D) なら真，そうでなければ偽
xor	排他的論理選言	(A=B) xor (C=D)	(A=B) または (C=D) の一方のみが成立すれば真，そうでなければ偽
or	論理選言	(A=B) or (C=D)	(A=B) または (C=D) の少なくとも一方が成立すれば真，そうでなければ偽
imp	論理内包	(A=B) imp (C=D)	(A=B) ならば (C=D) が成立すれば真，そうでなければ偽
eqv	論理同値	(A=B) eqv (C=D)	(A=B) と (C=D) が同値なら真，そうでなければ偽

例を見てみましょう。

例 3.6.

(A + 3) >= 10

は， 数値変数 A を含む論理式です。この論理式は A + 3 の値が 10 以上の時（即ち，A の値が 7 以上の時），真になり，A + 3 の値が 10 未満の時（即ち，A の値が 7 未満の時），偽になります。

もう 1 つ例を見てみましょう。

例 3.7.

(A >2) AND (B <=0)

は，数値変数 A，B を含む論理式です。この論理式は A が 2 より大きくて，かつ B が 0 以下の時，真になり，それ以外の時，偽になります。

3.4 関数

式を構成する要素として，関数もあります。tbasic には多くの関数がありますが，これらの関数については，基本的な関数の項で説明しますから，ここでは 2 つ例を挙げるだけにしましょう。

Sin(x) は 3 角関数のサイン関数ですが，この関数は引数 x および関数値 Sin(x) とも数値です。ですから，数値演算子と組み合わせて，例えば，

(Sin(x+2)/4)^2

のような式を作ることが出来ます。この式の値は，x の値に 1 を加えた値の サインを計算し，その値を 4 で割り，その結果を 2 乗した値です。

また，Str$(x) は x の数値を文字列として変換する関数です。ですから，x は数値を値にとる式で，Str$(x) は文字列を値に取る式になります。ですから，

```
"Sin(" + Str$(2) + ")=" + Str$(Sin(2))
```

は値を文字列にとる式です。この場合，実際のこの式の値は

```
Sin( 2)= 0.909297426825682
```

という文字列です。

以後の説明で，色々な式の例が実際に現われて来ますので，式の作り方の説明はこれで終わりにしましょう。関数や演算子を使って色々な式を作ることが出来ますが，基本的にはここで説明したことの組み合わせです。

まとめると，

> 式は定数と変数から始めて，演算子と関数をそれらの規則に従って作る

ということになります。

4　基本的なコマンド・文

BASIC プログラムでもかなり高度なプログラミング技法が用意されていて，それを使うと，かなり高度なプログラムを書くことも出来ます。しかし，そのためにはいくらかの経験とそれなりの知識が必要でしょう。元々 BASIC は入門用の言語として設計されていますから，特に高度なものを要求しないプログラムであれば，簡単に書くことが出来ます。そして勿論限界はありますが，それでかなりのことが出来ます。

ここでは，BASIC プログラムの中で使われる最も基本的なコマンドとそれを使ってできる文について説明をします。

ここで説明する文を機能によっていくつかに分類すると

- 出力文 ：計算結果等を出力　 Print 文
- 代入文 ：計算途中結果等を変数代入　 Let 文
- 入力文 ：計算のためのデータを入力　 Input 文
- 制御文 ：プログラムの流れを制御
  - For Next 文
  - If then else end if 文
  - While Wend 文
  - End 文
- 注釈文 ：人間のためのメモ Rem 文

となります。これらの文を活用するだけでかなりのことが出来ます。

まず最初に制御文のうち End 文を説明します。

4.1　End 文

制御文はプログラムの流れを制御する文です。BASIC では多くの制御文があり，それらを上手に使うと色々な処理を行うことが出来ます。その中で最も基本的で，使い方の簡単な文が End 文です。(End 文を制御文というのは大袈裟かもしれません。FullBASIC 規格では End はどの分類にも入らない単独の文として扱われています。)

End は英語で「終わり」を意味しますが，BASIC プログラムでも，

```
End
```

と書くだけで一つの文になり，プログラムの終わりを意味します。正確には，BASIC システムはプログラム実行中に End 文に出会うと，プログラムの実行を終了します。ですから，プログラムを終らせるところに End と書けば良いわけです。

FullBASIC では End は必ず主プログラムの最終行に一つだけに書くことになっていますが，Microsoft 系の BASIC ではどこに書いても，又幾つ書いても良いことになっています。tbasic もそれに習っています。しかし，あまり多くの End をプログラムの中に散在させるのは好ましくありません。プログラムの流れが分かりやすいように，出来れば 1 つにするよう工夫しましょう。End 文の例はこれから沢山でで来ますから，ここでは 1 つだけ例を見てみましょう。

次のプログラムは BASIC で書かれた最も簡単なプログラムです。

例 4.1 (もっとも簡単なプログラム).

```
End
```

このプログラムを実行すると，最初の行の文が End ですから，何もしないですぐに終了します。役に立ちませんが，しかし，これでも立派な BASIC のプログラムです。

End 文について少しの補足

Quick BASIC 等の Microsoft 系の言語では，End 文が無くてもプログラムの最後に来るとプログラムは終了します。tbasic もこれに習って，End 文が無くてもエラーにはなりません。End 文が無い場合，プログラムの最後まで実行しそれで終了します。

また，Visual BASIC では Form を閉じることなどによりプログラムを終了させ，基本的に End 文は必要ではありません。しかし，モジュールの中で，強制的にプログラムを終了させるものとして残っています。

4.2 Print 文

次に出力文について説明しましょう。
出力文は計算結果を出力するための文です。

BASIC での出力文は Print 文が基本です。

Print は印刷するの意味で，元々はプリンタに印字するの意味です。これは昔ディスプレイが一般的でない時のなごりで，現在では「ディスプレイに表示せよ」の意味で，tbasic では「実行画面に表示せよ」の意味になります。

```
Print 〜
```

の形で使います。ここで 〜は印字項目の並びですが，具体的には，式をカンマ「 ,」 やセミコロン「 ;」で区切って並べたものです。(式については「式とその構成要素」を参考にして下さい。)

例 4.2 (Print の簡単な例). 例えば

```
Print 3+4
```

は，正しい Print 文ですが，これは 「3+4 の結果 7 を実行画面に表示せよ。」という意味の文です。3+4 を表示するのではなく，その結果 7 を表示することに注意して下さい。

一般に

```
Print 式
```

は,「式の値を表示せよ。」ということを意味します。この式の値は文字列のこともあります。

例 4.3 (文字列の表示).

```
Print "これは Test です。"
```

は，これは Test です。を実行画面に表示します。前後の””が表示されないことに注意して下さい。表示されるのは式ではなくてその値（内容）です。

もう 1 つ例を見てみましょう。

例 4.4 (式の計算結果の表示).

```
Print A*B
```

を実行すると，例えば，　A の値が 3 で，B の値が 6 ならば，実行画面に，　18　と表示されます。この場合も表示されるのは式ではなくてその値です。

次に Print 文を実行した時の表示の位置について説明しましょう。

例 4.5 (表示と改行).

```
Print 3
Print 4
```

を実行すると，数値 3 と 4 は実行画面のどこに表示されるのでしょうか？これは,

```
3
4
```

と表示されます。

つまり，Print 文を実行すると，値を表示した後改行して次の行の先頭に次の表示位置を定めます。

では改行だけして，1 行空けるにはどうしたら良いでしょうか？それは何も表示しない Print 文，即ち

```
Print
```

を使えば良いのです。即ち　Print　のみでも 1 つの文になり，改行（1 行あける）を行ないます。ですから

例 4.6 (改行の利用).

```
Print 3
Print
Print 4
```

を実行すると，実行画面に

```
3

4
```

と 3 と 4 との間が 1 行空けられて表示されます。

場合によっては Print 文で表示したすぐ次から表示をしたいこともあります。このような時には,

```
Print 式
```

ではなくて

```
Print 式;
```

と最後にセミコロン「;」を付けます。これも Print 文となります。つまり

```
Print 式;
```

は式の値を実行画面に表示しますが，次の表示位置は最後に表示したもののすぐ次になります。

例を見てみましょう。

例 4.7 (非改行の利用)**.**

```
Print 3;
Print "+";
Print 4;
Print "=";
Print 3+4
```

を実行すると，

```
 3+ 4= 7
```

と表示されます。この文は

```
Print 3;"+";4;"=";3+4
```

と一つの文にしても同じことが実行されます。この場合，3; ”+”; 4; ”=”;3+4　が印字項目の並びになり，区切り記号が「;」です。

区切り記号としてカンマ「,」を使うと，少し違う表示になります。カンマ「,」は 20 文字区切りで画面を考えて，次の区切りから表示します。（この 20 文字は数値の精度から決まって，tbasic 独自の値です。）ですから，

例 4.8.

```
Print 1/2,1/3
Print 1/4,1/5
Print 1/6,1/7
```

は

```
 0.5                 0.333333333333333
 0.25                0.2
 0.166666666666667   0.142857142857143
```

のように表示されます。

カンマは小数点を含む数値等を表示する場合，表示位置を揃えるのに使います。

まとめると

Print 式は式の値をディスプレイに表示して改行する。
Print は特にしていない限り，次の表示位置は改行し次の行の先頭になる。
区切り文字セミコロン ; は次の表示を改行をせずに，すぐ次から表示する。
区切り文字カンマ, は次の表示を改行をせずに，次の表示区切りの先頭から表示する。

となります。

次に代入文の説明をしましょう。

4.3 Let 文

変数は定数を入れる箱でした。そしてその箱の中に定数を入れたり出したりする必要があります。それを行うのが代入文です。(変数については「式とその構成要素」を参考にして下さい。)

BASIC での代入文は Let 文です。

Let は英語で "Let A be C." (A を C とする。) という使い方をすることがあります。Let 文の Let はこの意味です。BASIC では

```
Let A = C
```

の形で用いて,「変数 A に式 C の値を代入する」という意味を持ちます。 正式には,

```
Let 変数 = 式
```

の形で使います。

ここで，変数が数値変数なら式の値は数値，変数が文字列変数なら式の値は文字列でなくてはいけません。

例を見てみましょう。

例 4.9 (単純な Let 文).

```
Let A = 3
```

を実行すると，数値変数 A に数値 3 が代入され，変数 A の値（内容）が 3 になります。

ですから,

```
Let A = 3
Print A
```

を実行すると，実行画面に 3 が表示されます。

別の例を見てみましょう。文

例 4.10 (式の値の代入文).

```
Let B = 3 * 5
```

を実行すると，数値変数 B に数値 15 が代入されます。

ですから，

```
Let B = 3 * 5
Print B
```

を実行すると，実行画面に 15 が表示されます。

別の例を見てみましょう。文

例 4.11 (変数を含む式の代入文).

```
Let B = 3 *A
```

を実行すると，数値変数 B に 3 掛ける A の値が代入されます。

ですから，

```
Let A = 3
Let B = 3 * A
Print B
```

を実行すると，実行画面に 9 が表示されます。

次に文字列変数の例を見てみましょう。文

例 4.12 (文字列の代入文).

```
Let P$ = "COMPUTER"
```

を実行すると文字列変数 P$ に文字列定数”COMPUTER”が代入されます。

ですから，

```
Let P$ = "COMPUTER"
Print P$
```

を実行すると，実行画面に COMPUTER と表示されます。

別の例を見てみましょう。文

例 4.13 (文字列式の代入文).

```
Let Q$ = P$ + "GAME"
```

を実行すると文字列変数 Q$ に文字列変数 P$ の内容と文字列定数”GAME”が結合されて代入されます。

ですから，

```
Let P$ ="Computer"
Let Q$ =P$ + "Game"
Print Q$
```

を実行すると，実行画面に Computer Game と表示されます。

まとめると

Let 変数 = 式
は式の値を変数に代入。
ここで式の値は，変数が数値変数なら数値，文字列変数なら文字列

となります。

Let 文での重要な補足があります。実は多くの BASIC では

Let 文で Let を省略しても良い

ことになっています。tbasic でも省略可能です。つまり，

```
Let P$ = "Computer"
Let Q$ = P$ + "Game"
Print Q$
```

の代わりに

```
P$ = "Computer"
Q$ = P$ + "Game"
Print Q$
```

と書いて良いわけです。そして実際のプログラムでは省略したものが殆どです。ですから，皆さんもプログラムを書くとき，Let を省略して書いて良いわけです。しかし，正式には Let 文であることは忘れないで下さい。

BASIC には = という論理演算子がありました。Let を省略した Let 文と論理演算子 = は形が良く似ています。BASIC を初めて勉強するとき，この２つはよく混同されます。くれぐれも注意して下さい。

Let を省略した Let 文を使って，少し具体的な例をあげましょう。

例 4.14 (よく理解する必要のある例)**.**

```
A = 1
A = A + 1
Print A
```

を実行したとすると，どのような結果になるでしょうか。

説明. 1 行目の A = 1 は数値変数 A に値 1 を代入したものですから，1 行目が終了時には A の値は 1 です。２行目の A = A + 1 は少し違和感のある式かもしれません。数学的には A = A + 1 は全く不可解な式ですが，ここでの = は等号ではないことを思い起こして下さい。つまり A = A + 1 は 数値変数 A に式 A + 1 の値を代入することでした。BASIC システムはまず 式 A + 1 の値を計算します。このとき， A の値は 1 ですから， A + 1 の値は 2 です。この値を数値変数 A に代入します。この時点で，A の値が 1 から 2 へ変化します。ですから，２行目 A = A + 1 が終了した時点で， A の値は 2 になります。ですから，３行目 Print A を実行すると， 2 が実行画面に表示されます。 □

似たような例をあげます。結果がどうなるか推測した後，実際にプログラムとして実行して確かめて見ましょう。

例 4.15 (結果を推測して，確かめよう).

```
A = 4
A = 3 * A
Print A
```

4.4 Input 文

次に入力文の説明をしましょう。

Let は変数に値を代入するものでしたが，それはプログラムの中で行なうものです。これに対してプログラムの実行中に，キーボードから所定の変数に値を代入する方法があります。それが Input 文です。つまり

キーボードからの入力には Input 文を使います。

Input 文字定数; 変数名

の形で用いて，文字定数をディスプレイ に表示した後，キーボードからの入力を待ち，リターンキーが押されると入力したデータを変数に代入を行ないます。

Input 文字列定数; 変数名

で文字列定数は入力するものを指示する言葉をいれます。これをプロンプト（入力促進メッセージ）と言います。プロンプトを付けない

Input 変数名

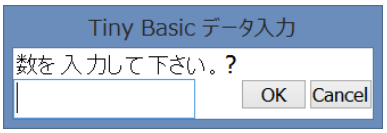
の形でも使えますが，入力の誤りをなくすために，どのようなものを入力するのか良く判る言葉を入れましょう。

例を見てみましょう。

例 4.16 (簡単な Input 文).

```
Input "数を入力して下さい。";A
```

を実行すると，ディスプレイの中央に下のような数値入力用のボックスが表示されます。



そして例えば， 5 を入力した後，OK ボタンをクリックすると，数値 5 が数値変数 A に代入されます。

次のプログラムを見てみましょう。

例 4.17 (Input 文を使ったプログラム).

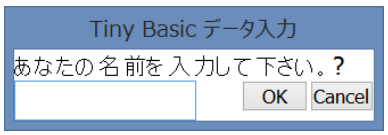
```
Input "数を入力して下さい。";A
Print A;
Print "の 2 乗=";
Print A *A
End
```

このプログラムを実行すると，「数を入力して下さい。」と表示された入力ボックスが現われ，キーボードからの入力を待ちます。そして例えば， 5 を入力した後，OK ボタンをクリックすると，数値 5 が数値変数 A に代入され， ディスプレイに 5 の 2 乗 = 25 と表示されます。

文字列の例を見てみましょう。文

```
Input "あなたの名前を入力して下さい。";N$
```

を実行すると，ディスプレイの中央に下のような文字列入力用のボックスが表示されます。



そして例えば， TARO を入力した後，OK ボタンをクリックすると，文字列定数"TARO"が文字列変数 N$ に代入されます。

次のプログラムを見てみましょう。

例 4.18 (文字列入力の例).

```
Input "あなたの名前を入力して下さい。";N$
Print "はじめまして";
Print N$;
Print "さん。"
End
```

このプログラムを実行すると，「あなたの名前を入力して下さい。」と表示された入力ボックスが現われ，キーボードからの入力を待ちます。そして例えば， TARO と入力した後，OK ボタンをクリックすると，文字列定数 TARO が文字列変数 N$ に代入され，実行画面に「はじめまして TARO さん。」 と表示されます。

まとめると

Input 文字定数; 変数名
は
入力ボックスに文字定数を表示し，
キーボードからの入力を待ち，
OK ボタンがクリックされると入力したデータを変数に代入する

となります。

Input ボックスでの入力について補足

入力完了を示すために，OK ボタンをクリックするのが標準ですが，この代わりに，Enter キーを押しても同じです。入力の場合こちらの方が便利です。何も入力しないで，OK ボタンあるいは Enter キーを押した場合，数値変数の場合は 0 が入力されたのと同じです。何も入力しないで，OK ボタンあるいは Enter キーを押した場合，文字列変数の場合は 空文字が入力されたことになります。

以上のことを使って簡単なプログラムを作ってみましょう。

例 4.19 (消費税の計算).

```
Input "金額"; Goukei
SouGoukei = Goukei * 1.08
Print "消費税込みの金額は";Sougoukei;" 円です。"
```

これは，金額を入力して，消費税を含んだ金額を表示するものです。このプログラムですと小数点以下の金額が出てしまいます。普通は四捨五入か切り捨てをします。まだ説明をしていませんが，BASIC にはこの四捨五入をする関数 Round と切り捨てをする関数 Fix があります。これを使うとより現実に即したものになります。End も追加しましょう。

```
’四捨五入での消費税
Input "金額"; Goukei
SouGoukei = Round(Goukei * 1.08)
Print "消費税込みの金額は";Sougoukei;" 円です。"
End
```

```
’切り捨てでの消費税
Input "金額"; Goukei
SouGoukei = Fix(Goukei * 1.08)
Print "消費税込みの金額は";Sougoukei;" 円です。"
End
```

このプログラム実際に書いて，実行して動作を確認しましょう。

次に制御文のうち，For Next 文の説明をしましょう。

4.5 For Next 文

計算機は，単純で膨大な計算をすばやく，そして正確に行なうのが得意です。このような計算の中に，同じ或は似たことを繰り返して，行なうというものがあります。BASIC でもこのような操作は簡単に出来ます。それを行なうのが For Next 文です。

For Next 文は具体的には

```
For 数値変数名=数値式（始まり値） to 数値式（終わり値）[step 数]
   文 1
   文 2
   …
Next 数値変数名
```

の形で使います。例を使って説明しましょう。

例 4.20 (For Next 文の基本形).

```
For I =1 to 5
   文 1
   文 2
   文 3
Next I
```

は，

I が 1 から 5 まで，文 1，文 2，文 3 を繰り返し実行する。

即ち，今の場合，文 1，文 2，文 3 を繰り返し 5 回実行することになります。

具体的に見てみましょう。プログラム

例 4.21 (For Next 文の基本的例).

```
For I =1 to 5
   Print 3;
   Print "の 2 乗=";
   Print 3*3
Next I
End
```

を実行すると，

```
Print 3;
Print "の 2 乗=";
Print 3*3
```

が 5 回実行されますから，実行画面に

```
3 の 2 乗= 9
3 の 2 乗= 9
3 の 2 乗= 9
3 の 2 乗= 9
3 の 2 乗= 9
```

と表示されます。

繰り返される文に変数が含まれる場合を見てみましょう。プログラム

例 4.22 (For Next 文の基本的例).

```
For I =1 to 6
   Print I;
   Print "の 2 乗=";
   Print I*I
   Next I
End
```

を実行すると，

```
Print I;
Print "の 2 乗=";
Print I*I
```

が 6 回実行されますが，I の値が 1 回ごとに変化します。ですから，実行画面に

```
 1 の 2 乗= 1
 2 の 2 乗= 4
 3 の 2 乗= 9
 4 の 2 乗= 16
 5 の 2 乗= 25
 6 の 2 乗= 36
```

と表示されます。

もう一つ別の例を見てみましょう。プログラム

例 4.23 (For Next 文の基本的例).

```
A = 0
For I =1 to 10
   A = A + I
   Print A
Next I
End
```

を実行すると，

```
A = A + I
Print A
```

が 10 回実行されますが，I の値が 1 回ごとに変化します。 又　A =A + I は A + I の値を A に代入するわけですから，これは A に I を加えることになります。ですから，実行画面に

```
 1
 3
 6
 10
 15
 21
 28
 36
 45
 55
```

と表示されます。つまり 1 から 10 までの和を求めた訳です。

まとめると

For 数値変数 = 始まり値 to 終わり値
　　文 1
　　文 2
　　…
Next 数値変数

は 数値変数が始まり値 から終わり値 まで　文 1, 文 2, … を繰り返すとなります。

最後に step の説明をします。

For to Next 文では step を指定しないと，変数は 1 づつ増加します。しかし場合によっては，この増加の量を指定したいこともあります。これを指定するのが step です。特に負数を step に指定すると減少させることができます。

例 4.24 (step -1 の利用例).

```
For i=5 to 1 step -1
   print i
Next i
End
```

を実行すると，画面に

```
 5
 4
 3
 2
 1
```

と表示されます。

今までのことを使った簡単なプログラムを挙げましょう。

次のプログラムは 10 個の数を入力してその合計と平均を表示するものです。

例 4.25 (10 個の数の合計と平均の計算).

```
Sum=0
Ave=0
For i = 1 to 10
   Input "数を入力して下さい。"; n
   Sum = Sum + n
Next i
Ave = Sum/10
Print "和  = ";Sum
Print "平均 = ";Ave
End
```

実行して動作を確認して下さい。

4.6 If then else end if 文

For 〜Next 文は単純な繰り返しでした。しかし，計算によっては途中で，色々な判断し，それに従って繰り返しの回数を決定しなければならないことがあります。又，いくつかの場合分けや，条件による処理方法の違いがある計算 もあります。この様に

> 条件によって分岐する場合に，
> If〜then〜else〜end if

を使います。

```
If 論理式 then
   文 1
   ...
Else
   文 2
   ...
End if
```

の形で使い，論理式が真の時，文 1... を実行し，そうでないとき，文 2... を実行します。又，else 以下は省略してもかまいません。この場合は，

```
If 論理式 then
   文 1
   ...
End if
```

の形で使い，論理式が真の時文 1 ... を実行します。

ここで，1 行目の then の後には何も書かないで改行しなければいけません。同様に， else と end if も 1 行にこれらだけ書きます。

また then の次の行から else の前の行の間には文を何行書いても構いません。また同様に else と end if の間にも文を何行書いても構いません。

If 文について少しの補足

今説明した If then else end if 文は 2 行以上に渡って書く文です。この他に 1 行だけで完結する，If then else 文があります。 tbasic でもこの構文を使うことが出来ます。しかし，この文はむしろ旧式の構文です。N88BASIC などの旧型の BASIC では 1 行の If 文しかありませんでした。この文は then と else の間に多くの文を書く場合不都合で，それを改良したのが上に説明した If then else end if 文です。

1 行の If 文は基本的に使う必要はありません。構文規則が少し違うので，混乱を避けるため，始めのうちは If then else end if 文だけを使うようにしましょう。現在 1 行の If 文を使う積極的な理由は，短く書けるというということだけです。

例を見てみましょう。プログラム

例 4.26 (If then else end if の基本形).

```
Print "1 + 1 はいくつですか？ "
Input "答えを入力して下さい。";A
If A = 2 then
   Print "御名答！"
Else
   Print "残念でした。"
End if
End
```

を実行すると，
「1 + 1 はいくつですか？」と実行画面に表示され，「 答えを入力して下さい。」 と表示された入力ボックスが現われ，キーボードからの入力を待ちます。
2 が入力されると，数値変数 A に 2 が代入され，論理式 A =2 が真になりますから，「 御名答！」と実行画面に表示されます。
2 以外を入力すると，論理式 A =2 は偽になりますから，「残念でした。」と実行画面に表示されます。

別の例を見てみましょう。プログラム

例 4.27 (少し複雑な条件式の If then end if 文).

```
Kosu = 0
For i = 10001 to 20000
   If (((i mod 3 = 0) or (i mod 7 = 0)) and (i mod 11 <> 0))  then
      Kosu = Kosu +1
   End if
Next i
Print Kosu
End
```

このプログラムは 10001 から 20000 までの間の自然数で，3 または 7 で割り切れ，かつ 11 で割り切れないものの個数を求めるものです。条件式が少しだけ複雑です。このような場合，括弧の付け間違いがよくありますから，注意しましょう。

これは単純な数え上げの問題で，理論的に計算する方法もありますが，この程度の個数の計算の場合は理論を使わず，素朴な方法で十分計算が出来ます。

tbasic では，数万回程度の計算は，大体すぐに計算が終わりますから，単純な条件比較だけで出来るものは素朴な方法で十分実用に耐えます。

まとめると

If 論理式 then
　　文 1
　　...
Else
　　文 2
　　...
End if
は，論理式が真の時，文 1 ... を実行し，そうでないとき文 2...
を実行します。else 部分は省略してもよい。

となります。

4.7 While Wend 文

BASIC で繰り返しを行うものとしては，For Next 文がありましたた。For Next 文は繰り返す回数が予め決まっているものについて実行するものです。しかし，繰り返しを行う場合，決まった回数ではなく，ある条件で繰り返す回数を決める必要があるものもあります。このような場合に使うのが， While Wend 文です。

While 〜 Wend で
While は 〜 している間の意味,
Wend は While End の略です。

While 〜 Wend は

```
While 論理式
   文 1
   文 2
   …
Wend
```

の形で用い，論理式が真の間，While と Wend の間の文 1，文 2，…の部分を繰り返し実行します。

例を挙げます。

例 4.28 (調和級数の和と項数).

```
S=0
I=0
While S < 10
   I = I + 1
   S = S + 1/I
Wend
Print I
```

この例は $1 + 1/2 + 1/3 + 1/4 + \cdots + 1/\mathtt{I}$ を計算して，この和が初めて 10 以上になる I を求めるものです。この級数は調和級数といって無限大に発散するものですから，このような I は確かにあります。しかし，それがどのような値であるかはすぐには分かりません。それを While Wend 文で計算しています。

条件 S < 10 の間繰り返しますから，和を取っていって S >= 10 になったら，繰り返しを止めて Wend の次の文に行きます。ですから，そのときの I の値が，和が 10 以上に初めてなったときのものです。

これも繰り返しですが，S = S +1/I の繰り返しが何回行われるか，予め分かりませんから， For Next 文で書くことは出来ません。

似た例をもう一つ挙げましょう

例 4.29.

```
Print "(1/N) < E となる N を求める"
Input  "E を入力して下さい。";E
N = 1
While (1/N)>=E
   N = N + 1
Wend
Print "(1/N) < ";E;" となる最小の自然数は";N;" です。"
```

これは，E を入力して，(1/N) < E となる最小の自然数を計算するものです。N が段々大きくなると，1/N はゼロに近づきますから，このような N は存在します。今度は (1/N) >= E の間， N = N + 1 を繰り返し，初めて (1/N) < E となったとき， Wend の次の文に進みます。

このように，繰り返す条件が式で簡単に書けるものは，While Wend 文が有効です。

このほか，入力データの制限をするときも While Wend 文は有効です。

例を見てみましょう。

例 4.30 (正数の入力).

```
Print "正数の入力"
While A<=0
   Input "正数を入力して下さい";A
Wend
Print A ;"が入力されました。"
End
```

この例では，正数が入力される迄，何回でも 入力の部分が繰り返されます。正数が入力されて初めて，Wend の次の文に進みます。正数を入力して欲しいときこのようにすると，間違いがありません。

For Next 文で挙げた例を少し改良してみましょう。

次のプログラムは 正数を入力してその合計と平均を表示するものです。

例 4.31 (いくつかの正数の総和と平均の計算).

```
Sum = 0
Ave = 0
Counter = 0
Input "数を入力して下さい。"; n
While n >0
   Sum = Sum + n
   Counter = Counter + 1
   Input "数を入力して下さい。"; n
Wend
If Counter > 0 then
   Ave = Sum/Counter
   Print "入力回数 = ";Counter
   Print "和  = ";Sum
   Print "平均 = ";Ave
End if
End
```

For Next 文の例では 10 個の数の入力でしたが，今回は回数は予め決まっていません。正の数をいくつでも入力して，その件数と和，平均を求めます。0 または負の数が入力されると，入力終了と解釈され，Wend の次の If 文に進みます。この If 文では Counter が正の場合，つまり実際に入力した場合，その入力回数と和，平均を計算して表示します。

実行して動作を確認して下さい。

もう一つ例を挙げましょう。

これは a と b を入力して最大公約数を計算するもので，ユークリッドの互除法といわれるものです。

例 4.32 (ユークリッドの互除法).

```
Print "a と b の最大公約数の計算"
Input "正整数 a = "; a
Input "正整数 b = "; b
While b> 0
   c = a mod b
   a = b
   b = c
Wend
Print "最大公約数 = ";a
End
```

プログラム自体は簡単ですが，これでなぜ最大公約数が計算できるかは少し説明が必要でしょう。しかし，ユークリッドの互除法については，プログラムの背景で述べますので，ここではプログラムを挙げるだけにします。

最大公約数の計算はユークリッドの互除法を使うのが普通ですが，tbasic では内蔵関数に最大公約数を与える関数 GCD があります。これを使うと，上のプログラムは

例 4.33 (GCD 関数の利用).

```
Print "a と b の最大公約数の計算"
Input "正整数 a = "; a
Input "正整数 b = "; b
Print "最大公約数 = ";GCD(a,b)
End
```

となります。

まとめると，While Wend は

```
While 論理式
   文 1
   …
Wend
```

の形で用い，論理式が真の間，While と Wend の間の 文 1… の部分を繰り返し実行する。

- For to Next 文 は繰り返しの回数が分かっているとき，
- While Wend は実行して繰り返しの回数が分かるとき使う。

となります。

4.8 Rem 文

プログラムは自分で作ったものでも，しばらく経った後から見ると，何を行なうものか判らなくなることがよくあります。また他人が作ったプログラムは，説明を聞かないとなかなか理解しにくいものです。そのためプログラムを作ると同時に，その説明書を作る必要があります。しかし，プログラムと別に説明書を作り，それを一緒に保管して置くことは，なかなか面倒なことです。

そこでプログラムの中にちょっとした説明を書いて置ければ，たいへん便利です。小さなプログラムではメモのような説明だけで充分なものもあります。この様な目的のためにあるのが　Rem 文　です。

Rem 文の Rem は Remark 即ち注釈の意味です。

```
Rem .......
```

の形で使います。

Rem の使い方

Rem の後の.....　の処には何を書いてもかまいません。BASIC システムは Rem に出会うと実行をすぐ次の行に移します。ですから Rem の後ろに何が書いてあっても，実行には関係がありません。そこで Rem の後にプログラムの説明等を書く訳です。この説明は日本語で書いても，英語で書いても入力できるものであれば，何でもかまいません。

更に Micorsoft 系の BASIC では Rem の代わりにコーテーションマーク ' を使うことも出来ます。tbasic でも使えます。また文の後に　:' を書いて，これ以後行末まで Rem 文とすることも出来ます。

Rem および' は 1 行 の注釈に使います。この方法だと複数行に渡るものはそれぞれの先頭に Rem を書く必要があります。tbasic ではこのために複数行に渡る注釈記号 { と } があります。{ と } が対になって注釈になります。tbasic では { があるとそれ以降 } まですべて文章を無視します。

例を見てみましょう。

例 4.34 (Rem の活用).

```
Rem    S =1 +2 +3 +… +N の計算
Print "S =1 +2 +3 +… +N の計算"
Input "N を入力して下さい。";N      :'N の入力
Let S =0                              :'S の初期設定
For I =1 to N
   S =S +I                            :'S に I を加える
Next I
Print S
End
```

この様に Rem 文を使って注釈を書いて置くと，プログラムの内容が良く判ります。Rem 文を上手に使って，判り易いプログラムにしましょう。但し，余り Rem 文を書きすぎるとかえって煩雑になってしまうことがありますので，程々にしましょう。

まとめると

行先頭に Rem (又は　') と書くと，この行は注釈で，実行に関係なく， 後に何を書いても良い。
文の後に:Rem (又は:') と書くと，以後行末まで注釈になる。
Rem 文を上手に使うよう心がけましょう。
tbasic ではブロック Rem として {　} が使える。

となります。

5　基本的な関数

BASIC プログラムでは基本的な文の他に，多くの関数を使うことが出来ます。ここではそれらのうち最も基本的なものについて説明します。個々の関数の詳しい使い方や，使える関数全体はマニュアルの関数の項目にありますので，そちらも適宜参照して下さい。

ここで説明する関数は，予め BASIC が持っている関数のことですが，BASIC ではユーザーが独自の関数を定義して使うことも出来ます。そのような関数をユーザー定義関数と言います。それに対して，予め BASIC が持っている関数のことを内蔵関数と言います。

tbasic では，関数は標準的な BASIC の関数は大体使えます。その他に tbasic 独自の関数もかなりあります。

基本的な内蔵関数を機能から分類すると次のようになります。

- 数値関数
- 数学関数
- 文字列関連関数

このほかに，tbasic 独自の

- 整数論的関数

や，ちょっと便利な

- 日付時刻処理関数

そして少し高度な

- ファイル処理系関数
- 入力処理関数

などがあります。

ここでは，数値関数，数学関数，文字列関連関数，整数論的関数，日付時刻処理関数から基本的なものについて説明します。

5.1　関数の使い方

まず，関数の一般的な形について説明します。関数は

```
Abc(x)
```

や

```
Fx$(s$,y)
```

などの形で使います。ここで，`Abc(x)` では `Abc` が関数名，`x` が引数（argument）です。引数が 1 つの場合 1 変数関数，２つの場合，２変数関数と言います。上の例で，`Fx$(s$,y)` は２つの引数 `s$` と `y` を持っています。引数の持たない関数もあります。

引数の場所には，普通マニュアルの説明では変数が書かれます。しかしここで表していることは，そこの位置に数値あるいは文字列が置かれるということです。ですから，変数でなくても値が同じ型であれば，式を置くことも出来ます。

例えば，`Abc(x)` と書かれているのは，`x` の位置に数値が置かれることを意味しています。ですから，実際のプログラムで `Abc(1)` と言った数値定数や，`Abc(x+y*z)` といったように，値が数値になる式 `x+y*z` を置くことも出来ます。

一般的には値が数値のものを数値関数，値が文字列のものを文字列関数と言います。また tbasic では値を論理値（真偽）に取る論理関数もあります。しかし，以下での分類は機能をもとにした分類で少し違います。

関数は，式を作るとき使います，式の作り方は「式とその構成要素」の項で説明しましたが，式の構成要素としての使い方は，変数や定数と同様です。例えば，Abc(x) という関数は，数値を値に取る関数とすると，Abc(x) を数値定数や数値変数と同様にして使えます。ですから Abc(x) を使って

```
(1+Abc(x)*3)/2
```

という式を作ることが出来ます。また，`Fx$(s$,y)` が文字列を値に取る関数とすると，

```
"Test"+Fx$(s$,y)
```

という式を作ることが出来ます。

5.2 数値関数

具体的な関数の説明をしましょう。まず数値関数について説明します。

> 数値関数は値を数値にとる関数のうち，数の基本的処理を行う関数です。

ここでは

(1) 絶対値 `Abs`
(2) 整数部分 `Int`
(3) 切り上げ `Ceil`
(4) 四捨五入 `Round`
(5) 符号 `Sgn`

について説明します。

(1) 絶対値 `Abs`

> `Abs` は Absolute Value (絶対値) の意味です。
>
> `Abs(X)`
>
> の形で使い，`X` の絶対値を与えます。

絶対値は，正数または 0 の場合，そのままの値を返し，負の数は符号を正にした値を返します。例えば

```
Abs(3.5)=3.5
Abs(0)=0
Abs(-3.5)=3.5
```

となります。

(2) 整数部分 Int

> Int は Integer (整数) の意味です。
>
> Int(X)
>
> の形で使い，X の整数部分を与えます。

Int(X) は，正確には X 以下の最大の整数を与えます。ガウス記号と同じです。ですから，X が負の時，注意が必要です。 例えば

```
Int(3.5)=3
Int(-3.5)=-4
```

となります。

(3) 切り上げ Ceil

> Ceil は上張りをする，或いは天井を張るといった意味ですが，ここでは整数へ切り上げをします。
>
> Ceil(X)
>
> の形で使い，X の小数部分を切り上げて整数にします。

Ceil は正の方向に切り上げをします。 ですから，例えば

```
Ceil(-2.8)=-2
Ceil(2.8)=3
Ceil(2)=2
Ceil(-1)=-1
```

となります。

(4) 四捨五入 Round

> Round は，丸めるの意味で，四捨五入して一番近い整数を返します。
>
> Round(N)
>
> の形で使い，N を四捨五入した値を返します。

Round(N) は N が負の場合は，絶対値を四捨五入したものに-1 を掛けたものを与えます。ですから，例えば

```
Round(0.5)=1
Round(-0.5)=-1
```

となります。

(5) 符号 Sgn

> Sgn は Sign (符号) の意味です。
>
> Sgn(X)
>
> の形で使い，X の符号を与えます。

Sgn(X) は，X の符号を与えますが，+ - で与えるわけではありません。

- X が正の時　Sgn(X)=1,
- X が 0 の時　Sgn(X)=0,
- X が負の時　Sgn(X)=-1

の様に符号を与えます。例えば,

```
Sgn(1)=1
Sgn(0)=0
Sgn(-2)=-1
```

となります。

5.3 数学関数

次に数学関数について説明しましょう。

> 数学関数は値を数値にとる関数のうち，数学でよく使われる関数です。

BASIC では数学で使われる多くの関数が使えます。主なものをあげると,

(1) 円周率 Pi
(2) ３角関数 Sin，Cos，Tan
(3) 平方根 Sqr
(4) 指数関数 Exp，対数関数 Log
(5) 最大値 Max，最小値 Min
(6) 乱数 Rnd

などです。この他に逆３角関数，双曲線関数，常用対数など使えます。

(1) 円周率 Pi

円周率 Pi は関数と言うより定数ですが,

```
Pi
```

という記号で表します。Pi は実際には 3.14159265358979 を表します。

(2) ３角関数 Sin, Cos, Tan

３角関数は数学で普通に使われていものです。

```
Sin(X)
Cos(X)
Tan(X)
```

は，それぞれ，X のサイン，コサイン，タンジェントを返します。初期設定では，角度は弧度法で表されますが，６０分法 を使うことも出来ます。

初期設定で使うと，弧度法で表され,

```
Sin(Pi/2)=1
Cos(Pi/4)=0.707106781186548
Tan(Pi/4)=1
```

のようになります。６０分法を使うには，Degrees コマンドを使います。例えば,

```
Degrees
Print Sin(45)
```

とすると，結果は サイン４５度

```
0.707106781186548
```

となります。弧度法に戻すには， Radians コマンドを使います。

３角関数ではこの他に，コタンジェント Cot，セカント Sec ，コセカント Csc が使えます。また逆３角関数として，アークタンジェント Atn，アークサイン ASin，アークコサイン ACos が使えます。

次のプログラムはサインとコサインの表を表示します。

```
Degrees
Print "Sin","Cos","角度"
For i=0 to 45 step 5
   Print Sin(i), Cos(i),i
Next i
End
```

(3) 平方根 Sqr

Sqr は平方根（Square Root）の意味です。

`Sqr(X)`

の形で使い， `X` の平方根を与えます。`X` が負の場合はエラーになります。

例えば

```
sqr(2)=1.4142135623731
sqr(10)=3.16227766016838
```

となります。`X^(1/2)` としても X の平方根が得られます。また，X の立方根は `X^(1/3)` で得られます。

次のプログラムは平方根と 3 乗根の表を与えます。

```
Print "平方根","３乗根"
For i= 1 to 5
   Print Sqr(i), i^(1/3), i
Next i
End
```

(4) 指数関数 Exp，対数関数 Log

指数関数は Exp です。

Exp は Exponential (指数) の意味です。

`Exp(X)`

の形で使い，自然体数の底を e (=2.71828 …) とするとき，e の `X` 乗を与えます。

例えば

```
exp(1) = 2.71828182845905 = e
exp(10)= 22026.4657948067
```

となります。

その逆関数が対数関数 Log(x) です。

Log は Logarithm（対数）の意味で,

`Log(X)`

の形で使い，X に対して，X =Exp(Y) となる Y を Log(X) として与えます。

例えば

```
Log(2)=0.693147180559945
Log(100)=4.60517018598809
```

となります。

Log(X) は自然対数 e を底にする対数（自然対数）です。常用対数 (10 を底にする対数) は Log10 で与えられます。また 2 を底とする対数は Log2 で与えられます。

次のプログラムは指数関数と対数関数の表を与えます。

```
Print "Exp","Log"
For i= 1 to 5
   Print Exp(i), Log(i), i
Next i
End
```

(5) 最大値 Max，最小値 Min

２つの数の大きい方または小さい方を指定するのが，最大値 Max， 最小値 Min です。

Max は Maximum（最大），Min は Minimum （最小）の意味です。最大値は

```
Max(x,y)
```

の形で使い，x と y のうち大きい方を与えます。最小値は

```
Min(x,y)
```

の形で使い，x と y のうち小さい方を与えます。

例えば，

```
Print Max(3,4)
Print Min(3,4)
```

を実行すると，

```
 4
 3
```

と表示されます。

(6) 乱数 Rnd

プログラムによっては乱数が必要なこともあります。BASIC では乱数を与える関数 Rnd が使えます。Rnd は Random Number （乱数）の意味です。乱数を関数というのは少し違和感がありますが，普通このように言われます。

乱数はいくつかの使い方がありますが，ここでは最も標準的な場合だけを説明します。

Rnd は Random Number （乱数）の意味です。

```
Rnd(1)
```

は，乱数を与えますす。 引数が同じでも違う数を返します。Rnd でも同じです。

例えば，

```
For i=1 to 10
   Print Rnd(1)
Next i
```

とすると，１０個の乱数が表示されます。

但し，乱数と言ってもコンピューターの内部では，ある方法によって計算されています。ですから厳密には乱数ではありません。乱数のような関数です[*3]。そして普通に上のプログラムを動作させると，１０個の異なった数を表示させますが，実行のたび毎回同じ１０個の数が表示されます。

実際に上のプログラムを２，３回実行してみて下さい。

これを避けるためには， Randomize コマンドを使います。Randomize コマンドは内部時計を使って乱数の初期化を行います。この文を上のプログラムに付け加えると，実行ごとに違った乱数が表示されます。

```
Randomize
For i=1 to 10
   Print Rnd(1)
Next i
```

実際にこのプログラムを２，３回実行して，上のプログラムとの違いを確認して下さい。

5.4 文字列関連関数

次に文字列関連関数について説明しましょう。

コンピューターは計算機といわれ，数の計算が主なように思われますが，コンピューターは数の計算だけでなく文字列に対する色々な処理も得意です。BASIC でワープロを作るのは速度的に無理がありますが，色々な文書処理を BASIC プログラムで行うことも出来ます。

また文書処理だけでなく，普通のプログラムでも，文字列に対する処理が必要なこともあります。BASIC ではそのような目的のために多くの文字列の処理に関連した関数が用意されています。ここではその主なものとして次を説明します。

(1) 文字列抽出関数 Left$, Mid$, Right$
(2) 数値・文字列変換 Str$, Val
(3) 文字列性質関数 Len, InStr
(4) 大文字・小文字変換 UCase$, LCase$

(1) 文字列抽出関数 Left$, Mid$, Right$

文字列の中からいくつかの文字列を抽出することが必要なこともあります。そのための関数が， Left$, Mid$, Right$ です。

[*3] このようなものを疑似乱数と言います。

Left$ は 左側から何文字かを抽出するときに使います。Left$ は

```
Left$(X$,I)
```

の形で使い，文字列 X$ の左から I 番目迄の文字列を与えます。

Mid$ は中間の文字列を抽出するときに使います。Mid$ は

```
Mid$(X$,I,J)
```

の形で使い，文字列 X$ の I 番目から J 個の文字からなる文字列を与えます。

また， Right$ は右側から何文字かを抽出するときに使います。Right$ は

```
Right$(X$,I)
```

の形で使い，文字列 X$ の右から I 番目迄の文字列を与えます。

例をあげましょう。

例 5.1 (文字列の抽出). A$="Computer" のとき,

```
Left$(A$,3) = "Com"
Mid$(A$,4,2) = "pu"
Left$(A$,2) = "er"
```

となります。

(2) 数値・文字列変換 Str$, Val

BASIC で扱う対象は数値と文字列でした。我々日常的には 1 という数字を見たとき，これは数なのか，数字なのか，厳密に区別して扱ってはいません。状況に応じで，数と見たり数字と見たりしています。また，1+2 というものを見たときも，式なのか文字列なのか日常的には区別しません。

一方，BASIC ではこの区別は厳密です。例えば

```
Print 1+2
```

とすると， 3 と表示されます。 1+2 と表示したい場合は

```
Print "1+2"
```

としなくてはなりません。

しかし，時には両方の意味に用いたいときもあります。BASIC では 数値と文字列は形式が違いますから，同じものを２つの意味に使うことは出来ませんが，それらを互いに変換することは出来ます。そのための関数が Str$ と Val です。

Str$ は String（文字列）の意味で，数値を文字列に変換するとき使います。

Str$(**数値**)

の形で使い，数値を文字列に変換します。

Str$ の逆変換をするのが，Val 関数です。Val は Value（値）の意味です。

Val(**文字列**)

の形で使い，文字列を対応する値に変換します。

例えば，

```
A$=Str$(12)
```

は A$=" 12" と同じです。このような数値定数の場合は， Str$ を使う理由はあまりありませんが，Str$(x) のように数値変数の場合など有効です。

また，

```
B=Val("12")
```

は B=12 と同じです。

Str$ と Val を使ったプログラムをあげましょう。

例 5.2 (4 乗数とその 4 乗数の下一桁).

```
Print "4 乗数","4*4 乗数","4 乗数の下一桁","4*4 乗数の下一桁"
For i=1 to 10
   A = i^4
   B$ = Right$(Str$(A),1)
   C = Val(B$)^4
   D$ = Right$(Str$(C),1)
   Print A,C,B$,D$
Next i
End
```

このプログラムは，４乗した数 A の下 1 桁を B$ として，それを更に数として，４乗した結果を表示するものです。結果を見ると表示された数の下 1 桁はすべて同じですね。

(3) 文字列性質関数 Len，InStr

文字列性質関数 Len，InStr は文字列の性質を調べるものです。

Len は Length(長さ) の意味で，文字列の長さを与えます。

```
Len(X$)
```

の形で使い，文字列 X$ の長さを与えます。

また，InStr は文字列の中にある文字列を探します。InStr は In String (文字列の中で) の意味で，

```
InStr(n,A$,B$)
```

の形で使い，文字列 A$ の n 番目からの文字から，文字列 B$ を探し，見つかればその位置を開始位置からの半角文字数で与えます。見つからない場合 0 を返します。

例えば

```
A$ = "tbasic"
B$ = "in"
s = InStr(1,A$,B$)
t = InStr(2,A$,B$)
u = InStr(3,A$,B$)
```

を実行すると，s = 2, t = 1, u = 0 となります。

(4) 大文字・小文字変換 UCase$，LCase$

大文字・小文字変換関数は文字を大文字にしたり，小文字にしたりします。

UCase は Upper Case（大文字活字箱）の意味です。

```
UCase$(A$)
```

の形で使い，文字列 A$ を大文字に変換します。全角文字にも対応しています。

例えば

```
A$=UCase$("This is a test")
```

とすると，A$="THIS IS A TEST" と同じことになります。

LCase は Lower Case（小文字活字箱）の意味です。

```
LCase$(A$)
```

の形で使い，文字列 A$ を小文字に変換します。全角文字にも対応しています。

例えば

```
A$=LCase$("This is a Test")
```

とすると，A$="this is a test" と同じことになります。

これらを使ったプログラムの例をあげましょう。少し複雑ですが，今まで説明した文と関数しか使っていません。復習として，プログラムを解析してみて下さい。

例 5.3 (英文の整形).

```
Input "英文を入力して下さい"; In$
N = Len(In$)
If Right$(In$,1)="." then
   N = N - 1
   In1$=Left$(In$,N)
Else
   In1$=In$
End if
StPt=1
EndPt=1
While StPt <= N
   EndPt = InStr(StPt,In1$," ")
   If EndPt = 0 then
      EndPt = N
   Else
      EndPt = EndPt + StPt - 1
   End if
   LenOfW = EndPt-StPt+1
   W$=Mid$(In1$,StPt, LenOfW)
   Out$ = Out$ + UCase$(Left$(W$,1))+LCase$(Right$(W$,LenOfW-1))
   StPt = EndPt+1
Wend
Out$=Out$+"."
Print Out$
End
```

このプログラムは入力した英文を空白で単語に切り分け，その先頭文字を大文字，他を小文字にして表示します。例えば入力ボックスに対して

```
tHis is tEst pROGram.
```

と入力すると[*4]

```
This Is Test Program.
```

と出力されます。

上のプログラムの仕組みについて説明します。説明のために行番号を付加します。

```
01 Input "英文を入力して下さい"; In$
02 N = Len(In$)
03 If Right$(In$,1)="." then
04    N = N - 1
05    In1$=Left$(In$,N)
06 Else
07    In1$=In$
08 End if
```

[*4] ここでは，Input 文を使って入力をしていますから，カンマを含む文を入力した場合は，カンマの手前までしか読み込めません。それを防ぐには，ここでは説明していませんが，Line Input 文を使うのがよいでしょう。

```
09 StPt=1
00 EndPt=1
11 While StPt <= N
12    EndPt = InStr(StPt,In1$," ")
13    If EndPt = 0 then
14       EndPt = N
15    Else
16       EndPt = EndPt + StPt - 1
17    End if
18    LenOfW = EndPt-StPt+1
19    W$=Mid$(In1$,StPt, LenOfW)
20    Out$ = Out$ + UCase$(Left$(W$,1))+LCase$(Right$(W$,LenOfW-1))
21    StPt = EndPt+1
22 Wend
23 Out$=Out$+"."
24 Print Out$
25 End
```

説明.

- 1 行で入力文字列を In$ とします。
- 2 行で入力文字列の長さを N とします。
- 3 行から 8 行は文字列最後のピリオド"." があればそれを取り除いた文字列を In1$としています。ピリオド"."がなければ，In$ を In1$ とします。以下，In1$ を入力文字列として調べます。
- StPt は単語の先頭位置，EndPt は単語の最後位置 (空白を含む) です。初期値はいずれも 1 です。
- 11 行から 22 行の間単語を抽出して，それを Out$ に付加していきます。
- 12 行では単語の先頭位置から，次の空白位置を探します。
- 見つかれば, 入力文字列での位置を EndPt とします。空白が無ければ，最後までが単語と判断します。
- これらの処理が，12 行から 17 行です。
- 18 行では単語の長さを,LenOfW とし，
- 19 行でその単語を W$ とします。
- 20 行ではその単語の先頭を大文字, 残りを小文字に変換し，それを既に処理済の Out$ に付加します。
- 21 行で次の単語の先頭を単語の次に設定して繰り返します。

□

5.5 整数論的関数

次に tbasic 独自の関数である整数論的関数について説明します。

> 整数論的関数は，高校の数学の範囲に出てくる，初等整数論的関数で，
> あると便利と思われるものを付け加えたものです。

これらの関数は標準的な BASIC にはありませんが，数式処理的な言語にはある標準的なものです。tbasic で現在使えるものは以下のもです。

(1) ２項係数 Binomial
(2) 階乗 Factorial
(3) 最大公約数 GCD，最小公倍数 LCM

(4) 最小素因数 PrimeFactor

です。

(1) 2項係数 Binomial

Binomial は　二項の意味です。ここでは Binomial Coefficient （二項係数）を意味します。

Binomial(n,r)

の形で使い，$(x+1)^n$ の二項展開の x^r の係数を与えます。

これは n 個のものから， r 個取る組み合わせの個数と同じです。Binomial の省略形として， Binom も使えます。例えば

```
Binomial(20,11)=167960
Binomial(32,15)=565722720
```

になります。精度の問題から，Binomial は大きな数では正確な値を与えませんが，概数は分かります。

(2) 階乗 Factorial

Factorial は　階乗の意味です。

Factorial(n)

の形で使い，n の階乗 $n!$ を与えます。

Factorial の省略形として， Fact も使えます。例えば

```
Factorial(9)=362880
Factorial(15)=1307674368000
```

になります。精度の問題から，Factorial は 19! までしか正確な値を与えませんが，それ以上の数でも概数は分かります。

(3) 最大公約数 GCD，最小公倍数 LCM

GCD は　Great Common Divisor （最大公約数）の意味です。

GCD(a,b)

の形で使い，整数 a と b の（正の）最大公約数を与えます。

例えば

```
GCD(9,15) = 3
Gcd(126582,127823) = 1241
```

になります。

LCM は　Least Common Multiple （最小公倍数）の意味です。

`LCM(a,b)`

の形で使い，整数 a と b の（正の）最小公倍数を与えます。

例えば

```
LCM(9,15) = 45
LCM(123,234) = 9594
```

になります。

(4) 最小素因数 PrimeFactor

Prime Factor は素因数の意味でが，ここでは最小素因数の意味です。

`PrimeFactor(n)`

の形で使い，n の最小素因数を与えます。PrimeFactor の省略形として， PFactor も使えます。

例えば

```
PrimeFactor(1234567) = 127
```

になります。

正成数 n が素数の場合は，その最少素因数は，n になります。ですから，

```
if n = PrimeFactor(n) then ...
```

により素数判定ができます。

PrimeFactor を使えば，整数の素因数分解は簡単にできます。

例 5.4 (素因数分解).

次のプログラムは素因数分解をします。

```
Input "自然数 n を入力して下さい。";n
If n > 0 then
   Print n
   Print " =";
   While n>1
      p = PrimeFactor(n)
      Print p;
      n = n / p
      If n > 1 then
         Print " *";
      End if
   Wend
   Print
End if
End
```

5.6 日付時刻処理関数

最後に日付時刻処理関数について説明します。日付時刻処理関数は多くの言語で使えますが，tbasic でも色々使えます。

日付時刻処理関数は文字通り，日付や時刻に関するもので，時間やカレンダーなどの処理の時に使います。ここでは以下のものを簡単に説明します。

(1) 日付関数 Date$
(2) 曜日関数 DayOfWeek
(3) 時刻関数 Time$
(4) 閏年関数 IsLeepYear
(5) 時間関数 GetHour
(6) 分関数 GetMinute
(7) 秒関数 GetSecond

(1) 日付関数 Date$

Date$ は現在の日付を読み出します。結果は"YYYY/MM/DD"の形の文字列です。ここで，

YYYY は 西暦 (0000-9999)
MM は 月 (01-12)
DD は 日 (01-31)

を表します。

例えば次は今日の日付を表示します。

```
Print Date$
```

(2) 曜日関数 DayOfWeek

DayOfWeek は曜日の意味です。

DayOfWeek(日付)

の形で使い，その日の曜日を返します。ここで日付は Date$ の返す形 "YYYY/MM/DD"の形の文字列です。曜日は日曜日が 1,・・・，土曜日が 7 です。

例えば，次は今日の曜日を表示します。

```
NDays$="日月火水木金土"
Print "今日は";Mid$(NDays$,DayOfWeek(Date$),1);"曜日です。"
```

(3) 時刻関数 Time$

Time$ は時刻の意味です。

Time$

の形で使い，現在の時刻を与えます。Time$ は 8 半角文字からなる文字列で，HH:MM:SS の形式をしています。ここで

HH は時間（00-23）
MM は分　（00-59）
SS は秒　（00-59）

を表します。

例えば

```
Print Time$
```

は HH:MM:SS の形で時間を表示します。

(4) 閏年関数 IsLeepYear

Is Leap Year はうるう年か？と言う意味です。

IsLeapYear(Y)

の形で使います。ここで Y は西暦を示す自然数です。その年がうるう年の場合， 真（True）を返します。そうでない場合偽（False）を返します。

例えば，次のプログラムはうるう年の判定をします。

```
Input "西暦年を入力して下さい。";y
If IsLeapYear(y) then
   Print y;"年はうるう年です。"
Else
   Print y;"年はうるう年ではありません。"
End if
```

(5) 時間関数 GetHour

Hour は時間の意味です。

GetHour

の形で使い，現在の時間を 0 から 23 で表します。

(6) 分関数 GetMinute

Minute は分の意味です。

```
GetMinute
```

の形で使い，現在の分を 0 から 59 で表します。

(7) 秒関数 GetSecond

Second は秒の意味です。

```
GetSecond
```

の形で使い，現在の秒を 0 から 59 で表します。

次は現在の時間を表示します。

```
Print "現在は";GetHour;"時";GetMinute;"分";GetSecond;"秒です。"
```

最後に日付処理関数を使ったプログラムの例をあげましょう。

例 5.5 (元旦からの日数計算).

次のプログラムは今日は元旦から何日目かを計算するものです。

```
' 今日は元旦から何日目
D$=Date$
Year = Val(Left$(D$,4))
Month = Val(Mid$(D$,6,2))
Day = Val(Right$(D$,2))
Counter = 0
MonthData$="312831303130313130313031" :' 各月の日数
For i=1 to Month -1
   CurMonth = Val(Mid$(Monthdata$,2*i-1,2))
   If (i = 2) and IsLeapYear(Year) then
      CurMonth = 29
   End if
   Counter = Counter + CurMonth
Next i
Counter = Counter + Day
Print "今日は元旦から";Counter;"日めです。"
End
```

6　入門 Basic プログラム例

このページは，tbasic でのプログラミングの勉強の為の例をあげます。ここにあるものは この章で説明した項目だけを使った，BASIC プログラミングでの基本的なものです。

以下では次の例を説明します。

- 基本技法：和や積の計算
- バーコードのチェックディジットの計算

6.1　基本技法：和や積の計算

ここでは，BASIC のプログラムを書く上での基本的な技法の内，和や積を計算する基本的方法を説明します。

[和の計算]
いくつかの数の和や積を計算することは色々な状況であります。例えば

```
1 + 2
```

や

```
3 * 4
```

と言うものです。この程度の計算であれば，そのまま式を書いて，

```
X=1+2
```

や

```
Y=3*4
```

のようにプログラムすれば良いでしょう しかし，例えば，1 から 20 までの和を計算するとしたら，どうしたら良いでしょうか。勿論，

```
A=1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20
```

というプログラムを書けば一応，計算は出来ます。
しかし同じような問題で，1 から 2000 までの和の計算をするとしたら，この方法ではもう出来ません。1 行に書ける式の長さには限界があるからです。しかし，それだけでなく，このような書き方は余りスマートとは言えません。何か別な効率的な方法は無いでしょうか。

この計算をソロバンで行うとすると，ソロバンの珠をはじいて順次足していくことになります。このようなことと同様なことが変数の代入を使うと可能です。
次の式を考えます。

```
A=A+i
```

この文は， 数値変数 A の値に i を加えたものを変数 A に代入するものですが，言い換えると， A に i を加えることに他なりません。この操作を何度も繰り返すと，和を計算することが出来ますつまり，

```
A=A+1
A=A+2
A=A+3
A=A+4
A=A+5
A=A+6
A=A+7
A=A+8
A=A+9
A=A+10
```

を行うと，A に 1 から 10 まで加えたことになります。ここで，1 から 10 まで加えた和を求める場合，スタート時点で A の値は 0 でないといけません。ですからその場合，

```
A=0
```

を最初に書いておく必要があります。これを変数 A の**初期化**と言います。

勿論，上の式のように 10 行を使って書くのはスマートではありませんし，多くの数を加える場合，現実的ではありません。

実は，このようなことを組織的に行う方法が BASIC にあります。それは For Next 文です。For Next 文を使うと，1 から 10 まで加えた和を求めるプログラムは次のように書けます。これは

```
A=0 :' 初期化
For i=1 to 10
   A = A + i
Next
```

と書くことが出来ます。これを使えば，1 から 2000 まで加えることも簡単です。

```
A=0
For i=1 to 2000
   A = A + i
Next
```

とすれば良いだけですね。

この方法は，1 から 100 まで加えるといった単純なものでなくても，同様な方法で可能です。一般に

```
S=a1 + a2 + a3 + a4 + … + an
```

のような級数の計算も全く同じ方法で可能です。この場合は，

```
S=0
For i=1 to 100
   S = S + ai
Next
```

のようなプログラムになります。

具体的な例をあげましょう。

■自然対数の底 e の計算

上の事柄の応用として，自然対数の底 e の値を求めてみましょう。e は高校の教科書にも載っている量ですが，普通は

$$e = \lim_{n\to\infty}(1+\frac{1}{n})^n$$

で定義されます。ここで，n は自然数です。この値を計算する方法として，次が思いつくかもしれません。

例 6.1 (定義からの e の計算).

```
For i=1 to 10
   n=10^i
   e = (1+1/n)^n
   Print e
Next i
End
```

しかし，このプログラムを実験してみると分かりますが，最初の内は予想通りの値が出ますが，i が大きくなるとおかしな値が出てきます。その結果，正確な値 $e = 2.71828182845904524\cdots$ を求めることはできません。その理由は，BASIC の数の精度の問題です[*5]。ですから，この定義式から，e の値を計算することは，理論としては可能ですが，実際は計算に余り適した式ではありません。

実は，e を計算するに適した式として，

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots + \frac{1}{n!} + \cdots$$

があります。この式を使って，e を計算するプログラムを書いてみましょう。

今の場合，上の説明での加えていく式を書くと，

$$e = e + \frac{1}{n!}$$

となります。ここで，$n!$ は n の階乗ですが tbasic には階乗関数 Factorial があります。これを使って書くと次のプログラムができます。

例 6.2 (級数展開を使った e の計算 1).

```
e=1
For n=1 to 20
   e = e + 1/Factorial(n)
Next n
Print e
End
```

ここでの初期化は，e=0 ではなく， e=1 となっていることに注意してください。

[*5] 実際はこの問題は BASIC の問題ではなく，普通に使われているプログラミング言語ではすべて同様な状況が起きます。仮に C でプログラミングをしても状況は同様です。

[積の計算]

今までは和の話でした。しかし，積についても全く同様な考えで計算することが出来ます。

例えば，階乗の計算

$$1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10 * 11 * 12 * 13 * 14 * 15$$

と順次 1 から 15 まで掛けた結果を計算するとします。(勿論 tbasic には 階乗を計算する内蔵関数がありますから，実際にこのようなプログラムを書く必要はありません。)

```
F=1*2*3*4*5*6*7*8*9*10*11*12*13*14*15
```

としても，可能ですが，和の場合と同様に For 文を使って書くと，

```
F=1
For i= 2 to 15
   F= F * i
Next i
End
```

と書くことができます。

先ほどの級数展開を使った e の計算の例では，tbasic 特有の関数 Factorial を使いました。しかし，BASIC によっては，階乗関数がないものもあります。その場合は，上のことを利用して，階乗関数を自前で書けば可能です。次のようにすれば良いでしょう。

例 6.3 (級数展開を使った e の計算 2).

```
e=1
F = 1
For n=1 to 20
   F=F*n
   e = e + 1/F
Next n
Print e
End
```

e の値は実際には内蔵関数 exp を使うと exp(1) と表されます。この値と上で計算した値を比較してみると，級数展開を使った e の計算 1,2 いずれも内蔵関数での値 $e = 2.71828182845904524\cdots$ と一致します。ですから，上の方法でかなり正確に e が計算できることが分かります。

階乗でなくとも，数の積は同様に書くことが出来ます。例えば奇数の積

```
P=1*3*5*7*9*11*13*15
```

は

```
F=1
For i= 2 to 8
   F= F * (2*i-1)
Next i
End
```

と書くことができます。

以上のことの応用としてウォリスの公式で π を計算してみましょう。

■ウォリスの公式から π の計算

π を計算する公式の一つとして次のウォリスの公式があります。

$$\frac{\pi}{2} = \frac{2}{1} * \frac{2}{3} * \frac{4}{3} * \frac{4}{5} * \frac{6}{5} * \frac{6}{7} * \cdots$$

この式から π を計算するプログラムを書いてみましょう。まず，ウォリスの公式を正確に式で書く必要があります。掛けていく項を w_i とすると，ウォリスの公式は

$$\frac{\pi}{2} = w_1 * w_2 * w_3 * w_4 * \cdots * w_i * \cdots$$

と表されます。ですから，ここで，w_i が求められれば，後は上で説明した積を計算できます。一見この w_i は計算が難しそうに感じますが，場合分けをすれば簡単に求めることが出来ます。

分子を a_i とし，分母を b_i とします。このとき，表を書いてみると，次のようになります。

i	1	2	3	4	5	6	7	8	...
a_i	2	2	4	4	6	6	8	8	...
b_i	1	3	3	5	5	7	7	9	...

このことから，

a_i は　i が奇数なら $a_i = i + 1$　であり，　i が偶数なら $a_i = i$

b_i は　i が奇数なら $b_i = i$　であり，　i が偶数なら $b_i = i + 1$

となることが分かります。ですから，

w_i は i が奇数なら $w_i = (i + 1)/i$ であり，i が偶数なら $w_i = i/(i + 1)$

となります。奇数・偶数は mod 演算子を使って判定できますから，一般項 を w と書くと

```
If (i mod 2 )= 1 then
   w = (i+1)/i
Else
   w = i/(i+1)
End if
```

で計算できます。従って例えば $n = 100$ までの積を計算するして π を計算するプログラムは次のようになります。

例 6.4 (Wallis の公式を使った π の計算 1).

```
n=100
P=1
For i=1 to n
   If (i mod 2)=1 then
      w = (i+1)/i
   Else
      w = i/(i+1)
   End if
   P = P * w
Next i
Print "π = ";2*P
End
```

上のように，If 文を使っての場合分けではなく，式で書きたいと思う人もいるかも知れません。$(-1)^i$ の性質を使うと，

$$w_i = \left(\frac{i}{i+1}\right)^{(-1)^i}$$

と表されることが分かります。このことから，上のプログラムは次のようにも書けます。

例 6.5 (Wallis の公式を使った π の計算 2).

```
n=100
P=1
For i=1 to n
   w = (i/(i+1))^((-1)^i)
   P = P * w
Next i
Print "π = ";2*P
End
```

上にあげたウォリスの公式は次のように 2 項まとめて書かれることもあります。

$$\frac{\pi}{2} = \left(\frac{2}{1} * \frac{2}{3}\right) * \left(\frac{4}{3} * \frac{4}{5}\right) * \left(\frac{6}{5} * \frac{6}{7}\right) * \cdots * \left(\frac{2i}{2i-1} * \frac{2i}{2i+1}\right) * \cdots$$

このようにすると，一般項が簡単に書けます。この 2 項まとめた項を v_i と表すことにすると，

$$v_i = \frac{2i}{2i-1} * \frac{2i}{2i+1}$$

で，

$$\frac{\pi}{2} = v_1 * v_2 * v_3 * v_4 * \cdots * v_i * \cdots$$

と表されます。この v_i の積として，$\frac{\pi}{2}$ を表し，これから π を計算するプログラムを書くのは簡単で，以下のようになります。

例 6.6 (Wallis の公式を使った π の計算 3).

```
n=50
P=1
For i=1 to n
   v = (4*i^2/((2*i-1)*(2*i+1)))
   P = P * v
Next i
Print "π = ";2*P
End
```

ここで，`n=50` となっていることに注意してください。Wallis の公式を使った π の計算 3 のプログラムは 2 項をまとめて計算しているので，項をかける回数は Wallis の公式を使った π の計算 1,2 のプログラムの半分になります。

これら 3 つのプログラムを比べて，どちらが良いかは，考えの分かれるところかもしれません。プログラム 3 が一番分かりやすいかもしれません。ただ，プログラム 3 はプログラム 1,2 の計算を 2 項ずつまとめて計算をしているので，同じ計算をしているわけではありません。

n を大きくして，実行してどの位正確に π が計算できるか試して下さい。

6.2 バーコードのチェックディジットの計算

身近によく利用されているバーコードでは，読み取りミスがないかを調べる**チェックディジット**というものを用いています。ここでは，このチェックディジットを計算するプログラムを書いてみましょう。

身近にある商品には以下のようなバーコードが表示されているのよく見かけます。

これは，JAN コードと言われるもので，レジでの価格の読み込みや色々な商品管理に利用されています。ここに示したものは標準バーコードで，13 個の数字で構成されています。バーコードリーダーは縦縞のバーの組み合わせから数字を読み取ります。それぞれのバーの下に表示されている数字は，バーコードで表される数字です。

数字は全部で 13 個あります。このうち，左 12 個が商品の情報を表します。最後の 1 桁はチェックディジットと言って，左 12 個の数字から計算されるある数です。

バーから 13 個の数字を読み取った後，左 12 個の数値からチェックディジットを計算し，その値が，バーから読み取った右端の数値と一致した場合，正しく読み取れたと判断します。そうでない場合，読み取れなかったとして，再度の読み込みを促します。

バーから数値を読み取るときに誤りが起きないようないくつかの工夫がなされていて，読み取りミスはほとんどありませんが，最終段階でのチェックとして，本来必要とされる以上の数値を 1 つ加えて，読み取りミスを防いでいます[*6]。

チェックディジットの計算方法は次の通りです。

チェックディジット

全部で 13 桁の数字の内，チェックディジットを除いた，12 桁の数字を左から

$$a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8\ a_9\ a_{10}\ a_{11}\ a_{12}$$

と表します。このとき，チェックディジット a_{13} は

$$a_1 + 3a_2 + a_3 + 3a_4 + a_5 + 3a_6 + a_7 + 3a_8 + a_9 + 3a_{10} + a_{11} + 3a_{12}$$

の 10 進 1 位の数を 10 から引いた残りの数の 10 進 1 位の数で計算されます。

例えば，左側 12 個の数字が

4 9 0 1 2 3 4 5 6 7 8 9

[*6] 本来必要とされる以上のものを冗長と言いますが，情報関係のシステムでは，安全性を高めるために，敢えて冗長のものを付加し検査や訂正に利用しています。これを冗長化と言います。このチェックディジットも一つの冗長化です。

の場合,

$$4+3\cdot 9+0+3\cdot 1+2+3\cdot 3+4+3\cdot 5+6+3\cdot 7+8+3\cdot 9=12\underline{6}$$

となるので，$a_{13}=10-6=4$ となります。

■プログラムの作成方針

(1) 入力
入力は，Input 文を使いましょう。数値を入力させるとして，BarNum を入力用の変数として,

```
Input "バーコードの左 12 桁の数を入力してください";BarNum
```

とすればよいでしょう。

(2) a_i の計算
12 桁の数 BarNum の左から i 桁目の数値 a_i を取り出す必要があります。これには多少数処理が必要ですが，商演算子 ¥ と剰余演算子 mod を使うことで，可能です。$a,b>0$ のとき，a ¥ b および a mod b はそれぞれ

$$a=qb+r\ \ (0\leqq r<b)$$

で一通りに決まる q と r のことです。特に，b = 10^i の形のときは，q は，a の下 i 桁を取り除いた数になり，r は，a の上 i 桁を取り除いた数になります。例えば,

$$a=123456,\ b=10^2\ \text{のとき},\ a¥b=1234\ \text{であり},\ a\ \text{mod}\ b=56$$

となります。12 桁数 BarNum の左から i 桁目は右から $12-i+1$ 桁目であることに注意します。まず，右から $12-i$ 以下を取り除くと，左から i 桁目までがえられ，その数の下一桁が a_i になります。即ち，a_i は次で求まります。

$$a_i=\ (\ \text{BarNum}\ ¥\ 10\hat{}(12\text{-i}))\ \text{mod}\ 10$$

(3) $a_1+3a_2+a_3+3a_4+a_5+3a_6+a_7+3a_8+a_9+3a_{10}+a_{11}+3a_{12}$ の計算。
前節の和の方法に従って For 文を使って計算すると，次が得られます。

```
b=0
For i=1 to 12
   a= (BarNum ¥ 10^(12-i)) mod 10
   If (i mod 2 = 1) Then
      c = 1
   Else
      c = 3
   End If
   b= b + c*a
Next i
```

(4) a_{13} の計算
定義から,

```
a13 = (10 - (b mod 10)) mod 10
```

となります。

■プログラム例

以上から，次のプログラムが得られます。

例 6.7 (バーコードのチェックディジットの計算 1).

```
Input "バーコードの左 12 桁の数を入力してください";BarNum
b=0
For i=1 to 12
   a= (BarNum ¥ 10^(12-i)) mod 10
   If (i mod 2 = 1) Then
      c = 1
   Else
      c = 3
   End If
   b= b + c*a
Next i
a13 = (10 - (b mod 10)) mod 10
Print "チェックディジットは "; a13
End
```

上のプログラムで，c を 1 つの式で書きたいと思う人もいるかもしれません。少し考えると，次で得られることが分かります。

```
   c = 2 + (-1)^i
```

これを使うと，上のプログラムは次のようにも書けます。

例 6.8 (バーコードのチェックディジットの計算 2).

```
Input "バーコードの左 12 桁の数を入力してください";BarNum
b=0
For i=1 to 12
   a= (BarNum ¥ 10^(12-i)) mod 10
   b= b + (2+(-1)^i) * a
Next i
a13 = (10 - (b mod 10)) mod 10
Print "チェックディジットは "; a13
End
```

■文字列入力によるプログラム

以上のプログラムでは，Input 文での入力は，数値変数への入力

```
Input "バーコードの左 12 桁の数を入力してください";BarNum
```

でした。Input 文での入力は，文字列変数を用いることも可能です。そして場合によっては，文字列変数を使った方が便利なこともあります。

次に，文字列変数を使ったプログラムを書いてみましょう。

(1) 入力

Input 文での文字列入力は，入力用変数として，文字列変数を利用するだけで，構文は，数値変数の場合と同じです。入力用の文字列変数を BarStr$としましょう。変数名の最後にある$が文字列であることを指定します。

```
Input "バーコードの左 12 桁の数字列を入力してください";BarStr$
```

とすればよいでしょう。

(2) a_i の計算

今度は長さ 12 の文字列 BarStr$ の左から i 桁目の文字を取り出し，それを数値 a_i と見なす必要があります。これには文字列処理関数のうち，部分文字列の抽出関数 Mid$ と数値への変換関数 Val を使うと簡単にできます。

Mid$(BarStr$,i,1) は，BarStr$ の左から i 番目の文字を取り出します。また，Val(A$) は，文字列 A$を A$で表される数値に変換すます。これを組み合わせれば

$$a_i = \texttt{Val(Mid\$(BarStr\$,i,1))}$$

で実現できます。a_i が求められれば，後は上のプログラムと同じです。

以上から，次のプログラムが得られます。

例 6.9 (バーコードのチェックディジットの計算 3).

```
Input "バーコードの左 12 桁の数字列を入力してください";BarStr$
b=0
For i=1 to 12
   b= b + (2+(-1)^i) * Val(Mid$(BarStr$,i,1))
Next i
a13 = (10 - (b mod 10)) mod 10
Print "チェックディジットは "; a13
End
```

Input 文での数値入力，文字列入力の２つの入力方法を見てきました。では，どちらの方法が良いでしょうか。一般的な回答としては，場合に依るということですが，私は次の基準で使い分けています。

Input 文での入力変数

- 単純な数の入力は，数値変数入力
- それ以外，空白やカンマを含まない単純な文字列は文字列変数
- 空白やカンマ等を含む文章は Line Input 文[*7] を使う。

バーコードでは数字の列が書かれています。その個々の文字が数を表すのは確かですが，全体としての 12 桁の並びは，数を表すというより，数字の並びというのが適切でしょう。ですからこの場合は，文字列とみる方が適切と言えるかもしれません。この場合，Mid$関数を使うのが自然で見やすいものになります。ただここの数値は数記号ではなく，数を表すので，Val 関数を使う必要があります。

全体としてみると，内容を一度理解した人にとっては，"バーコードのチェックディジットの計算 3"が分かりやすく良いものに思えるでしょう。

[*7] ここで，Line Input 文はまだ説明をしていませんが，Input 文での文字列入力はいくつかの制限があるということに注意してください。詳しくは，Help での Input 文の説明を参照してください。