

MSX-BACON(仮)

ユーザーズマニュアル

目次

| | |
|-------------------------------|----|
| 1. はじめに..... | 3 |
| 2. MSX-BASIC によるプログラミング..... | 4 |
| 2.1. MSX-BASIC のテキストエディタ..... | 4 |
| 2.2. 制御命令..... | 9 |
| 2.2.1. コロン(:)..... | 10 |
| 2.2.2. GOTO..... | 11 |
| 2.2.3. IF..... | 13 |
| 2.2.4. GOSUB/RETURN..... | 14 |
| 2.2.6. END..... | 16 |
| 2.2.7. コメント..... | 17 |
| 3. 変数と式..... | 18 |
| 3.1. 算術演算子..... | 21 |

1. はじめに

MSX-BACON は、MSX-BASIC の文法そのままの記述のプログラムをコンパイルして高速化する BASIC コンパイラです。

MSX-BASIC を「分かりやすい」「とっつきやすい」と感じる方々が多い中、一方で「動作が遅い」という声も多いです。MSX 誕生から 40 年が経過した 2023 年、MSX0 の登場により、再び MSX-BASIC に実用の可能性が出てきたわけですが、「動作が遅い」という部分をもう少し改善したいと思い、BASIC コンパイラの設計に取りかかりました。そのため、MSX0 から追加された IoT-BASIC もサポートしています。ある程度組み上がってきたところで、MSX の創造主である西和彦氏から「今後の MSX0/MSX3 だけでなく、これまでの MSX/MSX2/MSX2+/MSXturboR を見捨てないようにしたい。MegaRAM カートリッジを製造するので、それに対応させて欲しい。」という話が舞い込んできました。もちろん、私は快諾いたしました。

このような経緯で生まれた MSX-BACON ですので、MSX/MSX2/MSX2+/MSXturboR/MSX0/MSX3 に対応します。MSX3 に関しては、MSXturboR までと互換性のある MSXOS 管理下の部分で動作するもののみ対応です。Ta0X 管理下のプログラムは、Python/Cython/C/C++などをご利用下さい。

技術力に自信の無い方、技術にはあまり興味は無いが日常を少し便利にしたい方、技術力はあるがちょっとしたプログラムをササッと作りたい方、昔作った遅いプログラムを速くしたい方、等が主なターゲットとなります。プログラミングに自信の無い方は、まずは MSX-BASIC で組んで動くようになってから、MSX-BACON で高速化して快適に動くように調整する、といった組み方も出来ます。

一方で、ぬるぬる動くアクションゲームを作りたいとか、MSXとは思えない動きのゲームを作りたい等、MSX の限界を突き詰めるような使い方には向きません。そういうケースは、アセンブラでサイクルベースの最適化が必要ですので、アセンブラを使って下さい。MSX-BACON は万能ではありません。

MSX-BASIC の遅さを、ほんの少し改善。よく使う処理は簡単な記述で短時間で書ける。そんなプログラミング環境が、MSX-BACON です。

MSX-BACON には、「MSX-BASIC にこんな命令があったら良かったのに」と思う命令を、独自の命令として追加してあります。それらを使うと、MSX-BASIC では動作させることが出来なくなりますが、ある要件において簡潔に高速に動作する記述ができます。必要に応じてご利用下さい。

MSX/MSX2/MSX2+/MSXturboR/MSX0/MSX3 と全ての MSX に対応するソフトウェアには、特別なロゴの貼り付けが許可されるそうです。MSX-BACON では、その対応のための独自命令（機種判別など）を搭載して、全機種対応ソフトウェアをサポートします。

2. MSX-BASIC によるプログラミング

2.1. MSX-BASIC のテキストエディタ

1980 年代後半～1990 年代前半くらいのころに MSX を使っていて、MSX-BASIC の画面を見たことがない人はいないとは思いますが、MSX0 をきっかけに始めて MSX に触れる方や、昔の事なんてすっかり忘れてしまった方もおられると思いますので、まずは簡単に MSX-BASIC の使い方について簡単に触れておきます。

MSX-BASIC には、BASIC プログラムを入力するための専用のエディタが内蔵されています。本体の電源を入れると MSX-BASIC の入力待ちの状態になると思います。（※機種によっては、専用のメニューが表示される場合があります。MSXturboR であれば、内蔵ソフトウェアスイッチを OFF にして電源を入れれば OK。その他の機種でも、メニューの中から BASIC を起動する選択をすれば MSX-BASIC の入力待ちの状態になります。）MSX-BASIC の入力待ちの画面は、Fig.1 入力待ち画面のような表示になります。

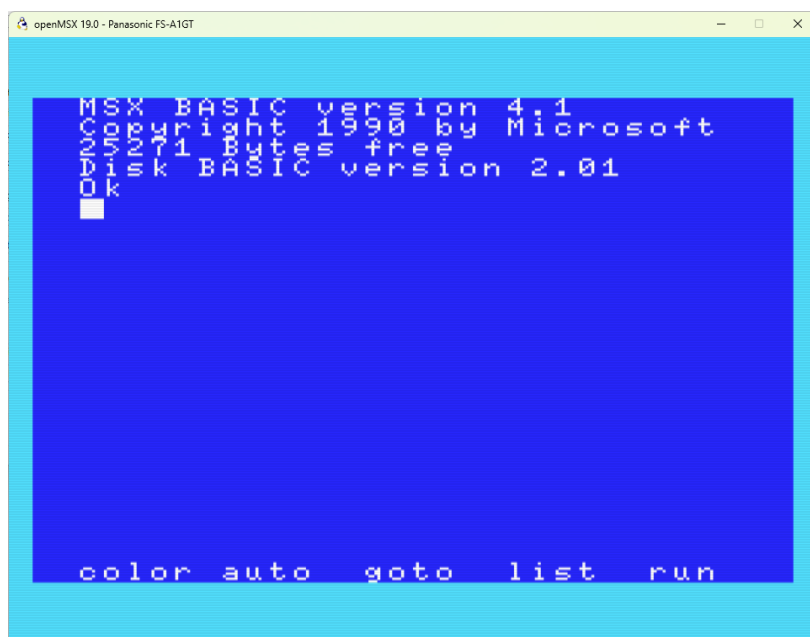


Fig.1 入力待ち画面

ここで PRINT “HELLO!” とタイプして [RETURN]キー（エミュレーターの場合、[Enter]キー）を押してください。Fig.2 HELLO!のような表示が出たと思います。

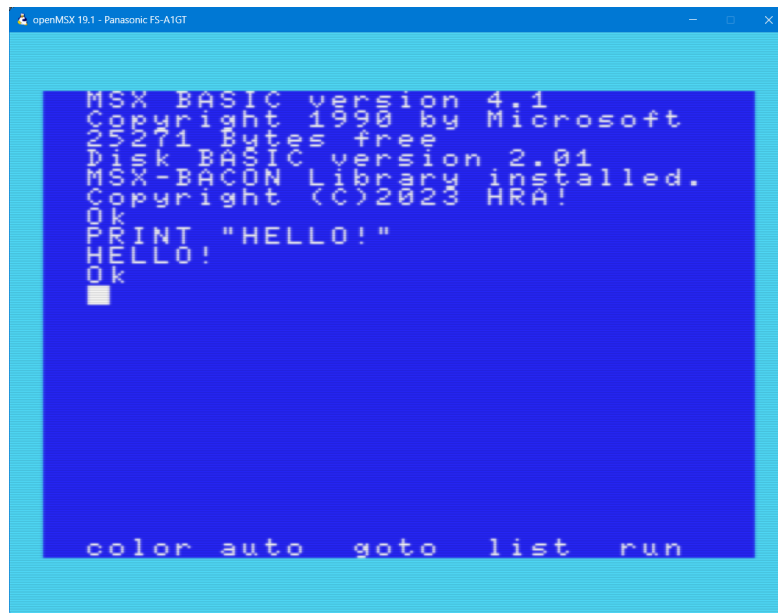


Fig.2 HELLO!

入力した PRINT が、「画面に指定の文字列を表示する命令」になります。BASICでは、この命令を並べていくことでプログラムを記述していきます。”HELLO!” が PRINT の第1引数で、「指定の文字列」になります。BASICでは、“ “（ダブルクォート）で囲んだ部分が文字列と解釈されます。

PRINT “HELLO!” の HELLO! を書き換えて [RETURN]キーを押すと、すぐ次の行に表示される内容も、書き換えた内容に変わることがわかると思います。

さて、[RETURN]キーを押すとすぐに実行されてしまいますが、どのように命令を羅列してプログラムに組み上げていくかを疑問に思った方もいるかもしれません。[RETURN]キーを押して即時に実行されるのを、今後「即時実行モード」と呼ぶことにします。それに対して、命令を並べてメモリにプログラムとして記憶させることを「プログラムする」と呼ぶことにします。

プログラムする方法ですが、先頭に行番号を記入します。試しに下記のように入力してください。1行入力するたびに、その行で[RETURN]キーを押してください。その行為でその行が記憶されます。

100 PRINT “HELLO!”

110 PRINT “WORLD”

画面は、Fig.3 プログラムを入力になっています。

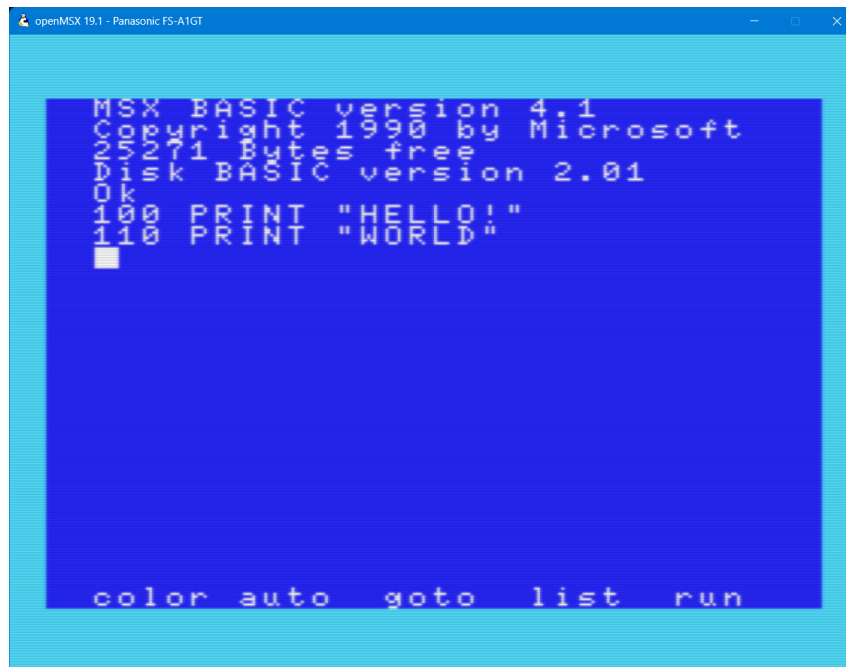


Fig.3 プログラムを入力

入力したプログラムが、正しく記憶されているか確認してみましょう。LIST と入力して [RETURN] キーを押してください。Fig.4 LIST 表示のように記憶されているプログラムが行番号昇順で表示されます。

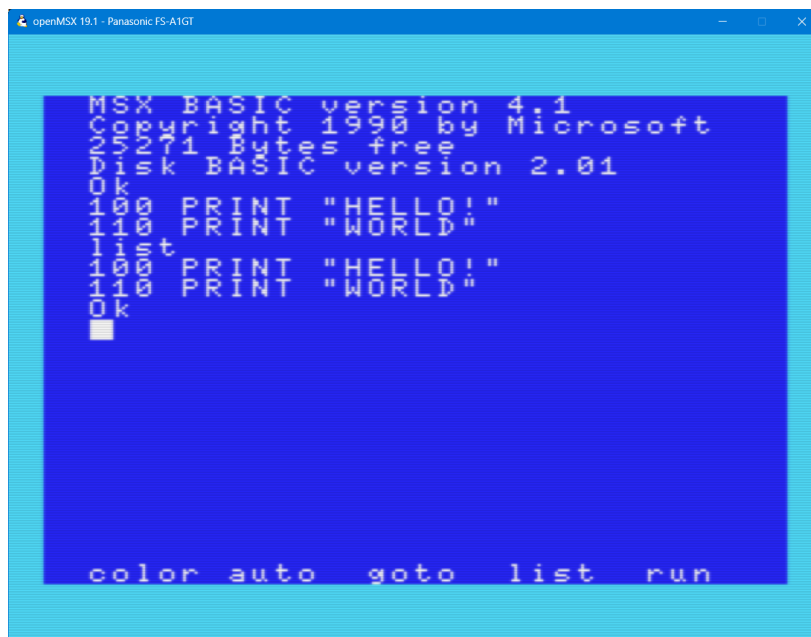
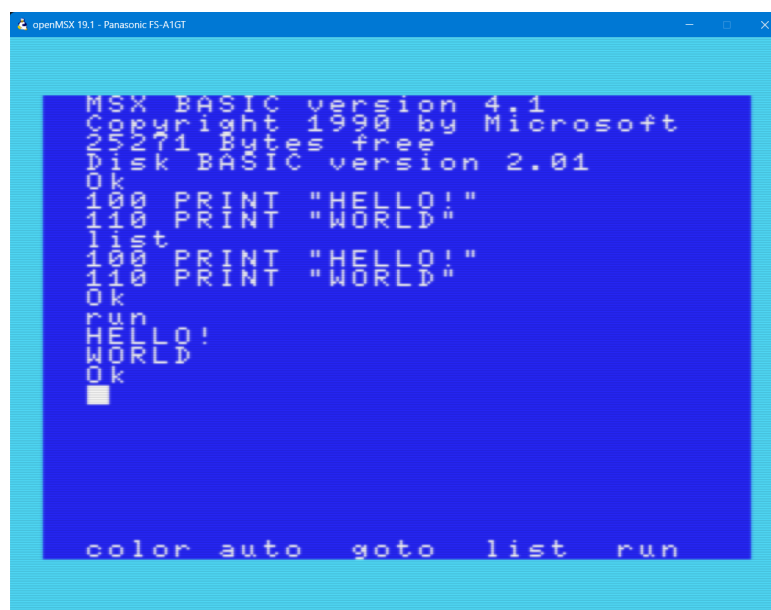


Fig.4 LIST 表示

LIST は、記憶したプログラムを表示する命令です。BASIC の命令は、アルファベットの大小に区別がありません。LIST のかわりに list でも、List でも、LiSt でも同じ動作をします。もちろん、PRINT も Print でも pRiNt でも同じ動作をします。ただし、プログラムとして記憶するのは、大文字に成形されます。例えば、100 print と入力しても、100 PRINT に置き換わります。

では、入力したプログラムを実行してみましょう。実行には、RUN 命令を即時実行モードで実行すると、そこから記憶されているプログラムが実行されます。では、RUN と入力して [RETURN] キーを押してください。

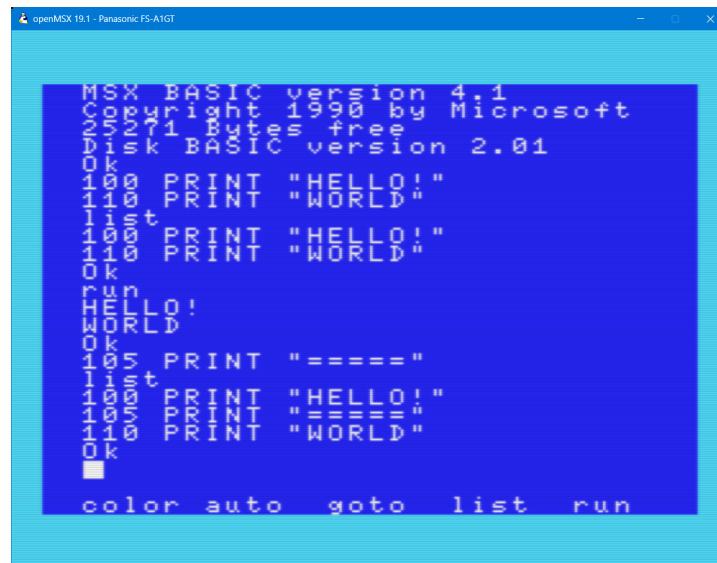
The image shows a screenshot of an MSX BASIC interpreter window titled "openMSX 19.1 - Panasonic FS-A1GT". The window has a blue background with white text. The text displayed is as follows:

```
MSX BASIC version 4.1
Copyright 1990 by Microsoft
255271 Bytes free
Disk BASIC version 2.01
Ok
100 PRINT "HELLO!"
110 PRINT "WORLD"
list
100 PRINT "HELLO!"
110 PRINT "WORLD"
Ok
run
HELLO!
WORLD
Ok
```

At the bottom of the window, there is a status bar with the text "color auto goto list run".

Fig.5 実行

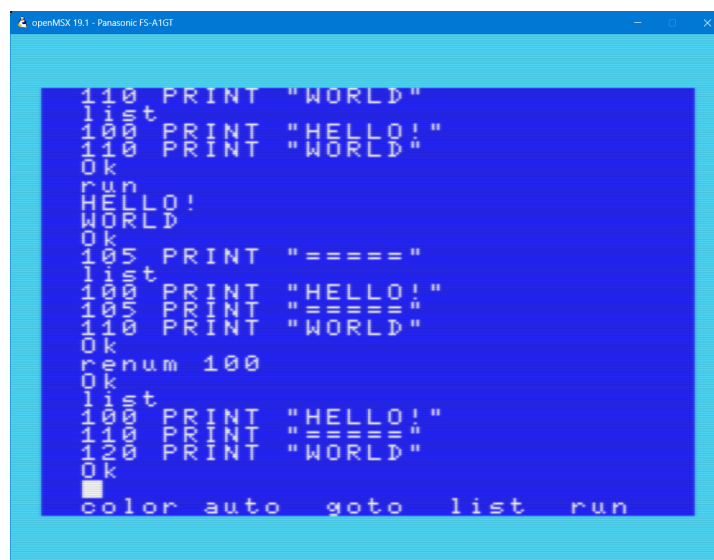
Fig.5 実行のように表示されたと思います。プログラムは、行番号の昇順に並べられて順番に実行されます。先ほどの例では、行 100 で HELLO! と表示し、行 110 で WORLD と表示します。PRINT 命令は、通常改行をつけるので、HELLO! を表示して改行した次の行に WORLD が表示されます。間の行番号を入力すると、自動的に行番号昇順になるように並べ替えられます。試しに、105 PRINT "====" と入力して [RETURN] キーを押してみてください。LIST すると、今入力した行 105 が、Fig.6 行の挿入のように行 100 と行 110 の間に入っているのを確認できます。このように、あとから追加しやすいように、行番号は 10 程度の間隔をあけて振っておくことをお勧めします。



```
MSX BASIC version 4.1
Copyright 1990 by Microsoft
25571 Bytes free
Disk BASIC version 2.01
Ok
100 PRINT "HELLO!"
110 PRINT "WORLD"
list
100 PRINT "HELLO!"
110 PRINT "WORLD"
Ok
run
HELLO!
WORLD
Ok
105 PRINT "====="
list
100 PRINT "HELLO!"
105 PRINT "====="
110 PRINT "WORLD"
Ok
color auto goto list run
```

Fig.6 行の挿入

挿入を繰り返して、行番号の並びが詰まり過ぎて、挿入するのに困ってしまった場合。RENUM 100 を実行してください。Fig.7 RENUM 100 の実行結果のように、行番号が振り直されます。この場合、自動的に 10 間隔になります。



```
110 PRINT "WORLD"
list
110 PRINT "HELLO!"
110 PRINT "WORLD"
Ok
run
HELLO!
WORLD
Ok
105 PRINT "====="
list
100 PRINT "HELLO!"
105 PRINT "====="
110 PRINT "WORLD"
Ok
renum 100
Ok
list
100 PRINT "HELLO!"
110 PRINT "====="
120 PRINT "WORLD"
Ok
color auto goto list run
```

Fig.7 RENUM 100 の実行結果

既に入力した行を削除したい場合、その行番号だけを入力して[RETURN]キーを押して下さい。

このように、行番号のついた命令を追加・修正・削除することによって、プログラムを入力していきます。

あとは、どのような命令があるのか、覚えていく必要がありますが、**すべての命令を覚える必要はありません。自分のやりたいことに必要な命令を優先的に覚えていけばいいのです。**その中でも、プログラムの流れを制御する命令は、どのようなプログラムを作るにせよ必要になる基本的な命令となりますので、まずはその制御命令を覚えてください。次章では、基本的な制御命令について説明します。

2.2. 制御命令

BASIC の制御命令はたくさんありますが、特に下記の制御命令はよく使う基本的な制御命令となります。これを覚えておけば、とりあえずどんなプログラムも組めます。**ここにはない制御命令は、ここにある制御命令の組み合わせでも実現できますので、無理して覚える必要はありません。**余力があれば覚えておくと「簡潔に記述できる」程度のもので、覚える命令を少なく済ませたい場合は、まずここに記載の命令を覚えてください。

- ・ :
- ・ GOTO
- ・ IF 条件式 THEN 文1 ELSE 文2
- ・ GOSUB
- ・ RETURN
- ・ END
- ・ コメント

では、以下で順に説明していきます。

2.2.1. コロン(:)

命令ではありませんが、まず：について説明しておきます。

複数の命令を覚えてくると、1 行 1 命令では右に隙間が出来て 1 画面に表示出来る情報量が減ってしまいます。隙間、もっと効率よく使いたいですよね？

そんなときに使うのが、：です。

隙間の効率化以外に、後述する IF 命令で、複数の命令を実行する場合には必須です。

：は、1 行に複数の命令を並べて記述するときに使う区切り記号です。命令の区切り目に：を記述しておけば、複数の命令を並べることが出来ます。

たとえば、このプログラム。

```
100 PRINT "HELLO!"  
110 PRINT "WORLD"
```

下記のように書き替えることが出来ます。

```
100 PRINT "HELLO!":PRINT "WORLD"
```

プログラムは見やすく書いておいた方が良いので、無理して詰める必要はありません。

しかし、スクロールしなければ見られないプログラムもまた見づらい。特に、MSX-BASIC はスクロールバーなどはなく、いちいち LIST しなければ見られないので、スクロールしない程度に詰めて書くことは、当時とても重要でした。

2.2.2. GOTO

GOTO 命令は、引数として行番号を指定します。指定の行番号へ制御を移します。

```
100 PRINT "HELLO!"  
110 PRINT "WORLD"
```

このようなプログラムを実行すると、先に HELLO! が表示されて、次に WORLD が表示されるのが分かります。これは、一番小さい行番号である行 100 を実行して、次に行 110 を実行しているわけですが、これを無限に続けたい場合、また行 100 へ戻ればいいわけです。

この「行 100 へ戻って下さい」と指示するのが GOTO 100 です。追加してみましょう。

```
100 PRINT "HELLO!"  
110 PRINT "WORLD"  
120 GOTO 100
```

これで、行 100 の「HELLO!を表示」を実行した後に、行 110 の「WORLDを表示」を実行して、その後に行 120 の「行 100 へ戻って下さい」を実行する。すると、行 100 へ戻るなので、また「HELLO!を表示」を実行し、「WORLDを表示」を実行し「行 100 へ戻って下さい」を実行する。これを無限に繰り返します。RUN して実行すると、Fig.8 GOTO サンプルの実行結果のようになるのを確認できます。

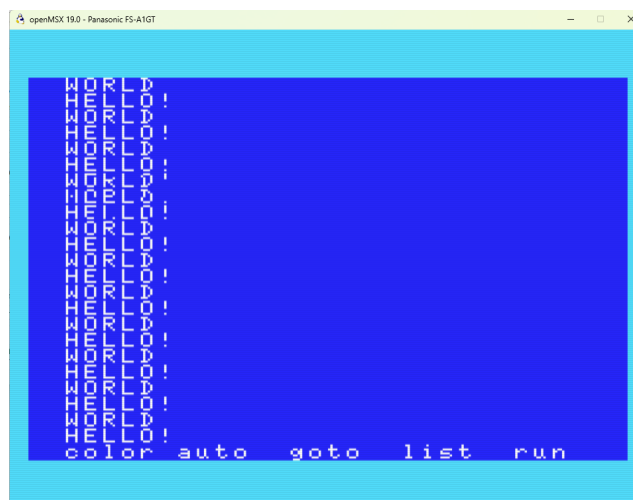


Fig.8 GOTO サンプルの実行結果

ちなみに、このように無限ループに陥った場合、[CTRL]キーを押しながら[STOP]キーを押す（以後、[CTRL]+[STOP]と記述します）と、Fig.9 [CTRL]+[STOP]で停止のように停止します。このサンプルのような無限に繰り返す処理を「無限ループ」と呼びます。

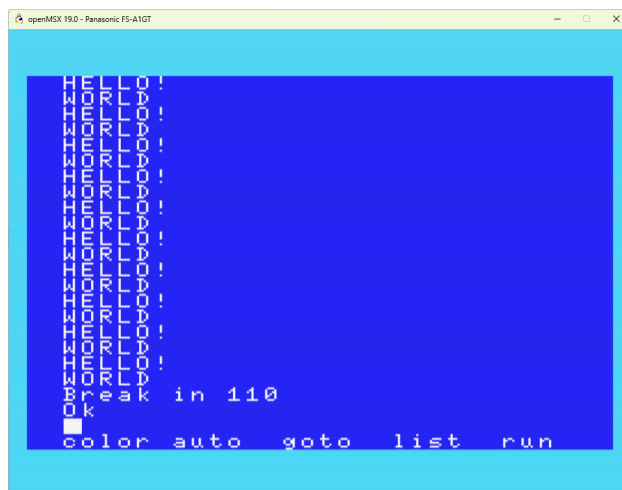


Fig.9 [CTRL]+[STOP]で停止

Break in 110 のように表示されますが、「行 110 を実行中に停止しました」という意味の表示です。行 100, 行 110, 行 120 があるので、止めるタイミングによって Break in 100 になるかもしれないし、Break in 120 になるかもしれません。

このサンプルプログラムのように簡単なプログラムの場合、ミスをするのではないかもしれませんが、複雑なプログラムを作ると、タイピングミスや勘違いなど様々な要因によって「意図しない動きをするプログラム」が出来上がることがあります。同じような表示をする箇所が複数合って、意図せず発生した無限ループが、一体何処で発生しているのか、表示だけではわかりにくい場合があります。そのような場合に、[CTRL]+[STOP]で止めて表示される Break in XXX の行番号 XXX は、問題箇所の追跡にとっても役立つヒントとなる事でしょう。

2.2.3. IF

GOTO で無限ループを作れることが分かったわけですが、状況によって動作を変えたいくなるケースに使うのが IF 命令です。IF 命令にはいくつかのバリエーションがあるのですが、最も基本的な IF 命令は下記のような書式となります。

```
IF 条件式 THEN 真で実行する文
```

条件式は式です。式の書き方については後述しますが、ここでは簡単に数式のようなものだと思って下さい。この条件式が 0 以外だった場合に「真 (True)」と呼びます。0 だった場合「偽 (False)」と呼びます。

THEN の右側にある「真で実行する文」とは、「命令」または、「: で繋げた複数の命令」を記述します。条件式が真の場合にのみ、「真で実行する文」が実行されます。

```
IF 1 THEN PRINT "HELLO!"
```

条件式に 1 と書いてあります。これは「0 以外」なので「真 (True)」です。従って、PRINT "HELLO!" は実行されます。

```
IF 0 THEN PRINT "WORLD"
```

条件式に 0 と書いてあります。これは「0」なので「偽 (False)」です。従って、PRINT "WORLD" は実行されません。

条件式が事前に真/偽が分かってしまう定数だと、IF 命令を使う意味は薄れてしまいましたが、後述する変数・式を使うことで、実行時に様々に変化する状況に応じて処理を切り替えることが出来る重要な制御命令となります。

2.2.4. GOSUB/RETURN

GOSUB は、サブルーチンコールと呼ばれるものです。GOTO と同じように行番号を引数に指定します。

| |
|-----------|
| GOSUB 行番号 |
|-----------|

「GOSUB 行番号」の記述で指定の行番号へ制御を移します。GOTO 行番号 との違いは、「GOSUB 行番号」の次の命令の場所を覚えて置いてくれることです。その場所に戻ってくるのが RETURN 命令です。

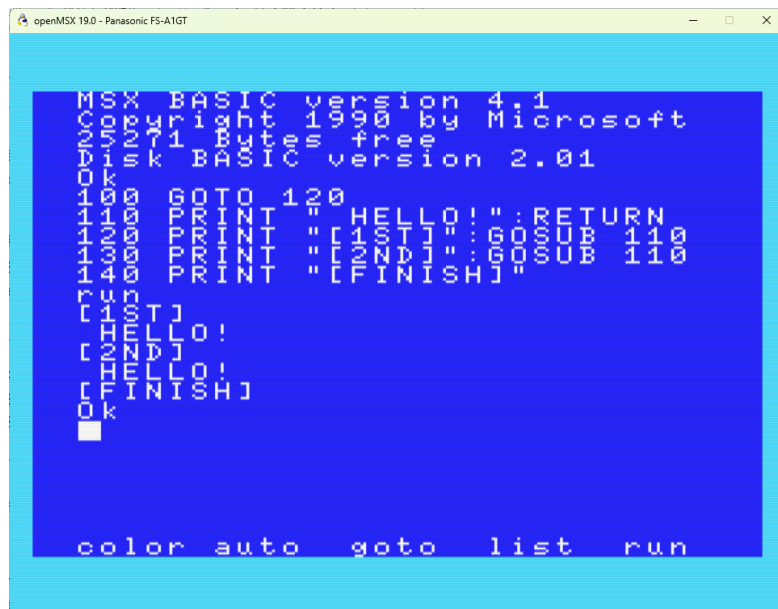
「GOTO で移動して、GOTO で移動先から次の場所へ戻ってくるのと何が違うの？」と疑問を持たれた方も居るかもしれません。GOSUB の飛び先が 1 回しか使われないのであれば、それで問題ありません。しかし、2 カ所以上から呼ばれる場合はどうでしょうか？下記のようなプログラムの動きを想像してみてください。

| |
|---|
| <pre>100 GOTO 120 110 PRINT " HELLO!":RETURN 120 PRINT "[1ST]":GOSUB 110 130 PRINT "[2ND]":GOSUB 110 140 PRINT "[FINISH]"</pre> |
|---|

行 110 に『「HELLO!」と表示する処理』があります。RETURN が付いているので、GOSUB を使って飛んでくる部分だと分かりますね。これを「サブルーチン」と呼びます。GOSUB は、「サブルーチンへ行け (GO SUBroutine)」の略なのです。サブルーチンへ飛んで実行することを「サブルーチンを呼ぶ」と言います。戻ってくるのが前提なので、呼ぶという言い方なのかもしれませんね。行 120 と行 130 の 2 カ所から GOSUB 110 で、行 110 のサブルーチンを呼んでいます。行 110 の処理を終えたら、行 110 の最後の RETURN で、元の場所へ戻るわけですね。

実行結果は Fig.10 GOSUB/RETURN のサンプルのようになります。

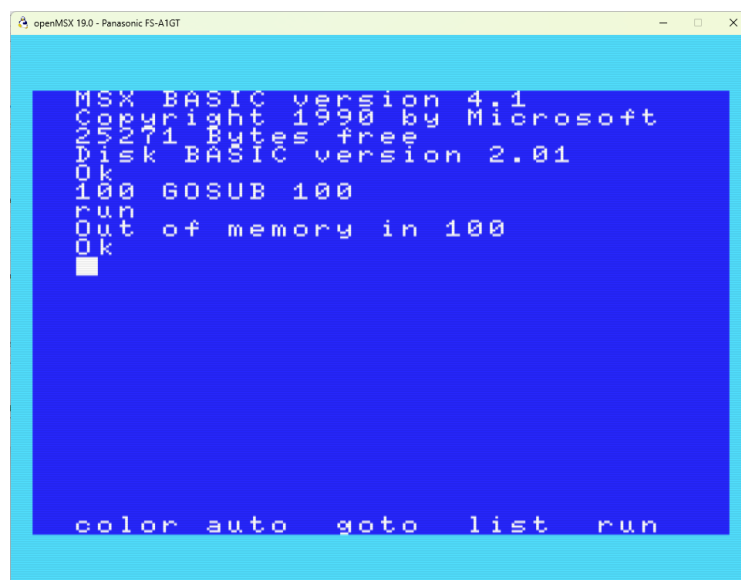
このサンプルは、サブルーチンを行 110 の 1 行にまとめましたが、サブルーチンは普通のプログラムなので、複数の行にまたがっても構いません。RETURN を実行した時点で元の場所に戻ります。



```
MSX BASIC version 4.1
Copyright 1990 by Microsoft
255271 Bytes free
Disk BASIC version 2.01
Ok
100 GOTO 120
110 PRINT "HELLO!":RETURN
120 PRINT "[18TT]":GOSUB 110
130 PRINT "[12ND]":GOSUB 110
140 PRINT "[FINISH]"
150 RETURN
160 PRINT "HELLO!"
170 PRINT "HELLO!"
180 PRINT "HELLO!"
190 PRINT "[FINISH]"
Ok
color auto goto list run
```

Fig.10 GOSUB/RETURN のサンプル

サブルーチンの中から別のサブルーチンを呼び出すことも出来ます。その場合、それぞれ RETURN で戻ります。GOSUB を実行すると、スタックメモリと呼ばれる場所に戻る場所を覚えておきます。RETURN は、その最新のものを調べて戻るわけですね。GOSUB を実行した回数 RETURN を実行すれば元に戻ってくるわけです。スタックメモリも有限(状況によって異なる)なので、何百回・何千回と GOSUB すると、スタックメモリが足りなくなってエラーになります。そのため、GOTO の代わりに GOSUB を使ってはいけません。何もせずに GOSUB だけを実行し続けるサンプル Fig.11 GOSUB による無限ループでメモリエラーを見ていただければ分かるように、エラーが発生します。GOTO と GOSUB は適切に使い分けて下さい。



```
MSX BASIC version 4.1
Copyright 1990 by Microsoft
255271 Bytes free
Disk BASIC version 2.01
Ok
100 GOSUB 100
run
Out of memory in 100
Ok
color auto goto list run
```

Fig.11 GOSUB による無限ループでメモリエラー

2.2.6. END

プログラムを明示的に終了させる命令が **END** です。これまで、プログラムを最後まで実行することで終了していました。Fig.10 GOSUB/RETURN のサンプルでも、サブルーチンを避けるために最初に **GOTO 120** してるのが美しくありません。**END** を使うことで下記のように書き直すことができます。

```
100 PRINT "[1ST]":GOSUB 1000
110 PRINT "[2ND]":GOSUB 1000
120 PRINT "[FINISH]"
130 END
1000 PRINT " HELLO!":RETURN
```

サブルーチンでないプログラムをメインプログラムと呼ぶことにします。考えながらプログラムを書いている場合、メインプログラムが何行になるか行 100 を書いている時点では分からないので、とりあえずサブルーチンは行 1000 くらいに置くことにしようと決め、**GOSUB 1000** と書いてしまいます。そして、メインプログラムの最後 120 の次に **END** を置くことでそこでメインプログラムを終了させます。サブルーチンは、その後に行 1000 として記述すれば良いのです。「この処理は何回も出てきそうだ、この処理のサブルーチンを作ろう、けど中身は後で考えよう、だから大きい行番号に配置することにしよう」と、思考の流れに合わせて書くことができます。

2.2.7. コメント

サブルーチンを沢山作っていると、どれがどれだか分からなくなることがあります。分からなくなったときに思い出す助けになるように、プログラム中に説明文を書くことも出来ます。

メモリを消費するので、あまり長いコメントは書けませんが、思い出すきっかけとなるキーワードや、メモをとっているならそのメモのページ番号等を記載しておくとう便利ですね。

REM 命令や、' 命令がコメントの記述です。タイピングは 'の方が楽（[SHIFT]+[7]）ですが、REMの方が省メモリになっていますので、REMの方をオススメします。REMと'の使い方は同じで、それより右側、行末までを「コメント」として扱い、プログラムとしては無視する動作となります。

```
100 PRINT "[1ST]":GOSUB 1010
110 PRINT "[2ND]":GOSUB 1010
120 PRINT "[FINISH]"
130 END
1000 REM -- Display 'HELLO!'. -----
1010 PRINT " HELLO!":RETURN
```

3. 変数と式

BASIC では、計算した結果を覚えておく事が出来ます。その覚えておく記憶場所に最大 2 文字までの名前を付けることが出来ます。名前のルールは、1 文字目はアルファベット。大小の区別はありません。2 文字目はアルファベットか数字。2 文字目もアルファベットの大小の区別はありません。3 文字以上の名前を記入してもエラーにはなりませんが、**先頭 2 文字しか認識されないのでご注意ください。**

記憶場所である変数へ、値を格納する行為を「値を代入する」と呼びます。=(イコール)の記号を使って、変数名=値の記述で値を代入します。下記では、A という変数に 100 という値を代入しています。

```
A=100
```

では、代入して表示するところまでやってみましょう。下記のように入力して [RETURN] キーを押してみてください。即時実行モードで実行されます。

```
A=100:PRINT A
```

実際に実行した結果が、Fig.12 変数への代入と表示です。

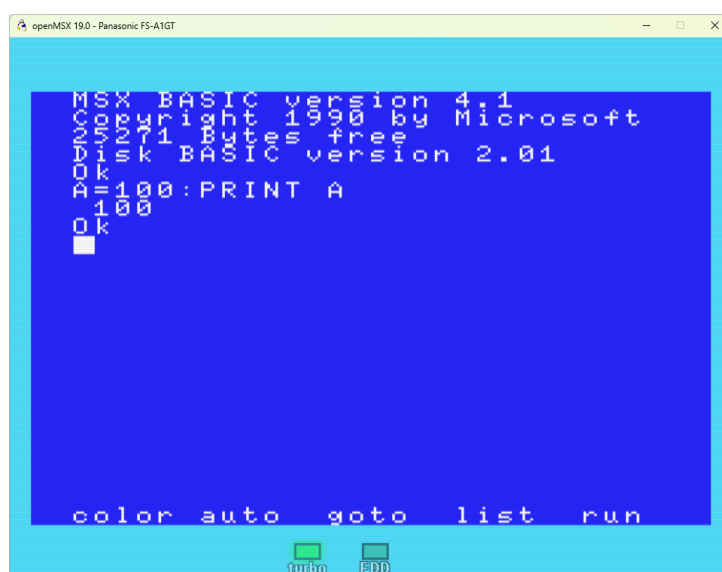


Fig.12 変数への代入と表示

ここで注意ですが、学校で習う数学では、「 $x=100$ 」とすると、「 x に 100 を代入する」と呼びますが、その一連の式の中では「 x は 100 を表現する別の記述法だ」といった意味になり、それ以降 x は全て 100 と等価になってしまいます。しかし、プログラムにおける代入は、あくまで変数という器に値を一時的に入れる行為に他なりません。 $A=100$ の後にすぐ $A=200$ を実行すれば、 A は 200 に変化するのです。後優先で、後から代入した値で前の値をキャンセル出来るのです。

$=$ の右側は、先ほど値と書きましたが、実は右側は式を記入する場所になっています。定数一つの記入は「式の最も簡単な場合」という扱いですね。式を記述するためには、 $+$ や $-$ といった演算子を使う必要があります。演算子には表 1 演算子のようなものがあります。

表 1 演算子

| 演算子記号 | 種類 | 意味 |
|-------------|-------|---------|
| $+$ | 算術演算子 | 加算、正の符号 |
| $-$ | 算術演算子 | 減算、負の符号 |
| $*$ | 算術演算子 | 乗算 |
| $/$ | 算術演算子 | 除算 |
| \wedge | 算術演算子 | 累乗 |
| \div | 算術演算子 | 整数の除算 |
| MOD | 算術演算子 | 剰余 |
| $<$ | 関係演算子 | 右が大きい |
| $>$ | 関係演算子 | 左が大きい |
| $<=$, $=<$ | 関係演算子 | 右が大きい一致 |
| $>=$, $=>$ | 関係演算子 | 左が大きい一致 |
| $<>$, $><$ | 関係演算子 | 不一致 |
| $=$ | 関係演算子 | 一致 |
| NOT | 論理演算子 | 否定 |
| AND | 論理演算子 | 論理積 |
| OR | 論理演算子 | 論理和 |
| XOR | 論理演算子 | 排他的論理和 |
| IMP | 論理演算子 | 包含 |
| EQV | 論理演算子 | 同値 |

演算子には優先順位があり、優先順位が高い演算子から先に計算されます。優先順位を表 2 演算子の優先順位に示します。

表 2 演算子の優先順位

| 優先順位 | 演算子 |
|------|---------------------------------|
| 1 | () で囲まれた部分 |
| 2 | 関数 |
| 3 | 累乗 |
| 4 | 符号 |
| 5 | *, / |
| 6 | ¥ |
| 7 | MOD |
| 8 | +, - |
| 9 | <, >, <=, =<, >=, =>, ><, <>, = |
| 10 | NOT |
| 11 | AND |
| 12 | OR |
| 13 | XOR |
| 14 | IMP |
| 15 | EQV |

同一の優先順位の演算子は左優先。() で囲まれた部分の中身は、また表 2 演算子の優先順位に従って計算されます。算術演算子については、数学と同じ優先順位です。それ以外の演算子について優先順位を覚えられない場合、記憶が曖昧な部分は明示的に () で囲うことで確実に期待の順番で処理されます。(※MSX-BASIC は () の分だけ遅くなりますので注意。)

3.1. 算術演算子

四則演算など、一般で使われる演算子です。除算の記号は、÷ではなく / です。分数の横線を斜めにしたイメージで覚えるとわかりやすいかもしれません。あるいは単位記号に現れる / と覚えると良いかもしれません（たとえば $[\text{km/h}]$ の / と同じ）。