PB－120
プログラミング入門講座

プログラムを簡単に作れたらいいな、と思ったことありませんか？
最近は学校教育にコンピュータが導入されているようですが、自分の好きなものを自分の好きなように作るというのは趣味の世界でしか成しえません.
私の場合、自分の好きなものがゲームだった訳ですが、作ったゲームで楽しむという目的が、いつしかプログラムを作ることそのものを楽しむようになってしまいました. その分できあがったゲームは他の人に楽しんでもらうことにしています.

1. 作ってみましょうプログラム

せっかくポケコンを持っているのなら、プログラムもやってみましょう。簡単ですから。
プログラムという言葉は知っていますよね。コンピュータを思い通りに操るためのものです。このプログラムはプログラム言語というものを使って作ることができます。
プログラム言語はコンピュータへ指示するための命令と手段のことで、PBの場合BASICという言語で指示を与えます。
概念だけ簡単に言ってしまうと、ポケコンという魔法の道具を操るためにBASICという魔法の言葉で呪文（プログラム）を紬ぐ、といったところでしょうか。とりあえず単純なゲームを作ってみましょう。

用意するのは、紙と鉛筆とPBです。紙と鉛筆はプログラムというよりも、ゲームを考えるときのアイデアスケッチや落書き、計算したいときのメモなどに使います。
プログラムは次の順番で作ります。といっても、この順番が絶対という訳ではありません。単なる目安、あるいは何でも予定通りにやらなければ気が済まない人用とでも思って下さればけっこうです。

(1)アイデアを考える.
(2)アイデアをPBの制約にあてはめる.
〈デザイン〉
(3)デザインをプログラムへ変換する.
(4)PBへプログラムを入力する.
(5)自分が気に入るまで直す.

2. アイデアがあれば大丈夫

まずどんなゲームにするか考えます。
RPGでしょうか？それともシューティング？まさか格闘ゲームなんて言いませんよね。別になんでもいいのですが、紙面の都合というものもあります。RPGの作り方はいずれ又の機会に、時間をかけてゆっくりとやりましょう。

さて、ここでは簡単なゲームにしたいので、ありがちではありますが数当てゲームにしましょう。

数当てゲームはPBが持っている「ある数」をプレイヤーが推理して言い当てるというものですが、何かヒントがなければ難しすぎて面白くありません。

そこで、プレイヤーが数字を言ったときに、PBはその中の合っている数字の数を表示することにします。

さあ、これがおおまかなアイデアです。次は、これをもう少し分かりやすく、また、プログラムにしやすいようにまとめてみましょう。

タイトル:数当てゲーム
内容:PBの持つ数をプレイヤーが当てるゲーム
ルール:
(1) PBは4桁の数字を隠している.
(2) プレイヤーは4桁の数字を推理して入力する.
(3) 入力された数字がはずれていた時、その中の当たっている数字の個数を表示する.
(4) プレイヤーが数字をすべて当てるまでゲームは続く.
(5) プレイヤーが数字を言い当てた時、それまでの入力回数を表示してゲームが終わる.

3.デザインはセンスがきめて？

工学社が発行しているポケコン・ジャーナル(通称PJ)という雑誌があります。携帯型情報端末の総合情報誌ということになっていたはずなのですが、その実ポケコン専門誌となっているのは編集部のせいか、はたまた読者と投稿者のせいか。ともあれ、そのPJクラブというページに時折「ゲーム・デザイン館」というものが登場します。

これは、1つのアイデアを元に多機種多人数でゲームを作り、そのできばえを競う企画なのですが、元は同じものだったはずのアイデアが、作る人の解釈のしかたでまったく違うゲームになることがあり、非常に楽しめる企画です。また、アイデアを出す人、作る人、遊ぶ人と、それぞれの段階で様々な人達が関わることができ、それぞれの人がプログラミングに参加できる利点もあります。

比較：
2536をPBが隠していた時、以下のように当たっているかどうかを比較します。

```
ＰＢ          ２    ５    ３    ６
              ↓違う↓違う↓同じ↓違う
プレイヤー    １    ２    ３    ４
```

同じが1つなので「HIT 1」

```
ＰＢ          ２    ５    ３    ６
              ↓違う↓違う↓違う↓違う
プレイヤー    ４    ３    ２    １
```

同じがひとつもないので「HIT 0」

画面：

```
1 : ?
```

[1][2][3][4]とキーを押す

```
1234_
```

[EXE]キーを押す

```
HIT 1 > 2:?
```

[4][3][2][1]を入力する

```
4321_
```

[EXE]キーを押す

```
HIT 0 > 3:?
```

[2][5][3][6][EXE]

```
ALL RIGHT!!
```

[EXE]キーを押す

```
COUNT 10
```

ここで、デザインと呼んでいるのは、その「ゲーム・デザイン館」のデザインです。先程出したアイデアを、プログラム可能なところまで詳細を決定していきます。

とはいえ、ここからはPBのBASICを知らないとできない作業なので、とりあえず、こういうふうにアイデアからデザインを作ると思っていただくだけでけっこうです。

4.いわゆるところのプログラミング

さあ、ここからが狭い意味でのプログラミングです。今までの部分もプログラミングという作業の一部なのですが、趣味レベルのプログラムの場合は、ここまでの部分を頭の中で済ましてしまい、いきなりBASICでリストを作ってしまうことが多いのも事実です。

デザインが固まったら、次はそれをBASICのプログラムに置き換えていきます。

BASICのプログラムは、主にデータを処理することを目的としています。ゲームでもそれは同じです。したがって、プログラムを作るためにはデータのことも考えなければなりません。といっても、慣れれば、プログラムとデータが同時に頭に浮かぶようになるので、普通の人はあまり意識していない事でしょう。

PBの場合、扱えるデータの種類は文字と数値の2種類しかありません（コラム1参照）。

数値というのは、123のように数字で表わされる、計算に使用する数のことです。数値を記憶させたり保存したりするためには数値変数というものを用います。数値変数はAからZまでのアルファベット1文字で表わされ、「A+1」のように計算に用いることができます。

文字というのは、アルファベットや記号等のことで、"Y/N?"のようにダブルクォーテーション（"のこと）で囲んで表現します。文字の記憶には文字変数というものを用います。文字変数はAからZまでのアルファベット1文字に$という記号を付けて表現され、「B$+"!"」のように文字データの加工に用いることができます。

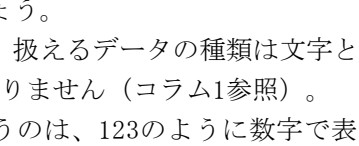
今回必要なデータは、PBが持っている数、プレイヤーの入力する数、入力回数の3つです。

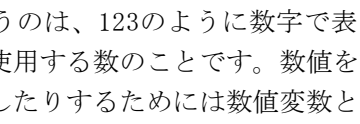
本文中にあるとおり、文字データはダブルクォーテーション(")で囲って表現し、数値データは数字で表現します。

また、文字データは文字変数へ、数値データは数値変数へ、それぞれ保存します。

その際、同じアルファベットで表現される文字変数と数値変数とは、同じ場所に記憶を行うため、1つの変数にはどちらか片方のデータだけしか記憶させることができません。これは、PBに特有の変数の構造なので注意が必要です。

詳しくは、マニュアルの「変数」のページ(PB－120入門書の場合P40)を参照してください。





コラム1　文字データと数値データ

PBが持っているのは4桁の数字です。これをデータとして持つためにはどうすれば良いのでしょうか？

数字なので数値でも文字でもどちらでも良いのですが、ここでは文字データとして扱うことにします。文字データの場合、後で入力した値と見比べるときに簡単に処理できるようになります(コラム2参照)。

プレイヤーの入力する数は、PBが持っている数と比較するためのものなので、これも文字データとします。

入力回数は、何回の入力で答えを言い当てることができたかを知らせるためのスコアの代わりになるものです。これは、回数を数えるためのものなので、数値データでなければなりません。

使用する変数:

PBが持っている数	：文字データ	→	B$
プレイヤーの入力する数	：文字データ	→	D$
入力回数	：数値データ	→	C

人間が数字の羅列を見比べる場合、人間にとって数字は単なる「数字」でしかないため、比較対象が文字の場合と同様のやり方になります。

```
１　２　３　４
↓同じ↓同じ↓違う↓違う
１　２　４　３
```

PBが比較を行う場合、比較対象が文字データのときと、数値データのときとで方法が異なります。

＊文字の場合

文字列の左側から1文字ずつ取り出して比較します。

```
"1234" → "1","2","3","4"

"1243" → "1","2","4","3"
```

```
"1"　"2"　"3"　"4"
↓同じ ↓同じ ↓違う ↓違う
"1"　"2"　"4"　"3"
```

＊数値の場合

計算によって各桁の値を取り出して比較します。

```
INT(1234        /1000)=1
INT((1234-1000)/ 100)=2
INT((1234-1200)/  10)=3
     1234-1230       =4

INT(1243        /1000)=1
INT((1243-1000)/ 100)=2
INT((1243-1200)/  10)=4
     1243-1240       =3
```

```
１　２　３　４
↓同じ　↓同じ　↓違う　↓違う
１　２　４　３
```

(INTはカッコの中の値を越えない最大の整数を返す関数で、正の値のときは小数点以下を切り捨てた整数となります)。

コラム2　データを見比べる

5.アルゴリズムはひものよう？

使用するデータとその格納場所(変数のことです)が決まったので、次はそのデータを処理する部分を作成します。

データを処理するそのやり方をアルゴリズムといいます。たとえば「このキヤラのアルゴリズムは単純で動きがミエミエだよ」などとフカシをいれるのに使われることもあります。まあそれはジョークとして、実際には「素数を求めるアルゴリズム」や「迷路データを自動生成するアルゴリズム」などのように、何らかの結果を得るためのやり方のことを言います。

さて、今回のプログラムが必要とするアルゴリズムは何でしょう？

ひとつは4桁の任意の数字を作り出すこと、もうひとつは入力された数字が当たっているかどうかを調べること、この2点が今回のポイントです。

PBの場合、任意の数値を作り出すためのコマンドとして「RAN#」があります。このRAN#は0から1の間の乱数(でたらめな数)を作り出してくれます。

これを利用すると4桁の数字が作れそうです。RAN#で乱数を4回出して文字変数B$に入れればよいのです。

しかし、RAN#の作り出す値は数値であって文字ではありません。とはいえ、数値を文字に変換するSTR$というコマンドがあるため問題にはなりません。それよりもやっかいなのはRAN#をどうやって0から9までの値にするかです。

これは、常当手段として「INT(RAN#*10)」という式が使われます。RAN#が0から1までの値を均一の割合で出力すると仮定して、その値を10倍して整数部を取り出すことにより0から9の値を作り出しています(コラム3参照)。

難しいことを書いてしまいましたが、ともかくこの式を使えば0から9までの1桁の整数を得ることができます。数字が1つ作れたら、後は同様に4桁作ってしまえばOKです。簡単ですね。

次は、入力された数字とPBが隠している数字とが同じかどうかを判定する部分を考えましょう。

文字列の一部分が等しいかどうかを調べるためには「MID」というコマンドを使用します。MIDは文字列の任意の位置から指定した個数の文字を抜き出すことができます。

本文中にはRAN#の値を0から1までの値と呼んでいますが、実際にはPBの有効桁数の範囲で出力されるため以下のような値をとることになります。

```
RAN#          →  RAN#*10     →INT(RAN#*10)
0.000000000   →  0.00000000  →  0
      :              :           :
0.100000000   →  1.00000000  →  1
      :              :           :
0.200000000   →  2.00000000  →  2
      :              :           :
0.300000000   →  3.00000000  →  3
      :              :           :
0.400000000   →  4.00000000  →  4
      :              :           :
0.500000000   →  5.00000000  →  5
      :              :           :
0.600000000   →  6.00000000  →  6
      :              :           :
0.700000000   →  7.00000000  →  7
      :              :           :
0.800000000   →  8.00000000  →  8
      :              :           :
0.900000000   →  9.00000000  →  9
```

コラム3　0から9の数字の作成

ただし、この便利なコマンドは専用文字変数$に対してのみ使用可能なので、一度$に文字列を入れてやらなければなりません。

やり方としては、比較したい文字列を専用文字変数$へ入れ、そこから比較したい文字を1文字ずつ取り出して同じかどうかをチェックしていきます。

文章にすると複雑ですが、実際のプログラムでは単純な形で表現できます(図1参照)。

二つの文字列を比べるときには、IF命令で単純に同じかどうかという比較をおこなうことができます。

```
IF B$=D$;･･･
```

この場合、文字列全部を比較の対象としてしまうため、1文字違っただけでも「B$≠D$」と判定されてしまいます。
1文字ずつ判定するためには、MID関数を使用して文字列を分解しなければなりません。

```
B$="1234"
D$="1243"

$ = B$+D$ = "12341243"

MID(1,1)  →  "2"  ←同じ→  MID(5,1)  →  "2"
MID(2,1)  →  "2"  ←同じ→  MID(6,1)  →  "2"
MID(3,1)  →  "3"  ←違う→  MID(7,1)  →  "4"
MID(4,1)  →  "4"  ←違う→  MID(8,1)  →  "3"
```

図1 文字列を1文字ずつ比較する

```
 10 BS=""
 20 FOR I=1 TO 4
 30 A=INT(RAN#*10)
 40 B$=B$+STR$(A)
 50 NEXT I
 60 C=0
 70 C=C+1
 80 PRINT C;":";
 90 INPUT D$
100 IF LEN(D$)≠4 THEN 80
110 $=B$+DS:E=0
120 FOR I=1 TO 4
130 IF MID(I,1)=MID(I+4,1);E=E+1
140 NEXT I
150 PRINT "HIT";E;" >";
160 IF E≠4 THEN 70
170 PRINT "OK!"
180 PRINT "ALL RIGHT!!"
190 PRINT "COUNT":C
```

リスト 数当てゲーム

6.おじやまな虫にはご注意を

プログラムを作っていると、しばしば自分の思い通りの動作にならないことがあります。

この予想外の動作、特に悪影響を及ぼす動作のことをバグといいます。虫のことですね。

このバグという言葉はなかなか一般にも定着しているようで、雑誌のミスプリなどをバグと呼ぶ人がちらほらみられます。この場合、単純に間違いという意味で使われるようです。

さて、プログラムのバグというと、ゲームなどではちょくちょく見ることができます。魔界村(カプコン)でタイムアウトぎりぎりにわざと呪文を受けてカエルになると、タイマが狂って時間制限がなくなる、などというのもプログラムのバグです(古いゲームですが)。この場合は、単なる間違いではなく、プログラマの意図しない動作またはその部分のプログラム自身をさしてバグと呼びます。

バグはプログラムの動作をぶち壊すばかりでなく、ひどい時にはハードにまで被害を及ぼすと言われています。まあ、PBの場合、BASICのプログラムが暴走しても[AC]キーで必ず停止でき、ハードどころかプログラム自身を壊すこともありません。安心してプログラムを作ることができます。

この辺が、いかにもPBというべきところで、入門用ポケコンとして良くできているところのひとつです。

もっとも、バグで何も壊れないとはいえ、変数に入っているデータは、プログラムしだいで変化してしまうかもしれません。それに、例えば一回入力するたびにエラーが発生するようなプログラムなんて、使いたくないものですよね。

やはり、バグはとってしまわなげればなりません。

バグのあるプログラムを修正して正しく動くようにすることをデバッグといいます。バグ取り、虫を退治するなど人によっていろいろな言い回しがありますが、全般に害虫を駆除するようなニュアンスのことばが使われていることと思います。つまり、悪いのは虫のせいだからそれをつぶしてしまおうというわけです。

ひとくちにバグといっても、大別すると2つに分けることができます。実行時にエラーが発生して停止してしまうものと、異常な動作のまま動き続けるものです。

エラーが出るものとして代表的なのは、リストを入力する時に押すキーを間違えて、実行時にERR2が出るパターンです。

エラーが出るバグの場合、多くはリストを眺めていれば、「あっ、何でこんなことしてんだ？」と思わず言ってしまうような部分がみつかるものです。こちらは、慣れればそれほど問題にはなりません。問題なのはエラーの出ないバグです。

異常な動作のまま動き続けるバグとしては、計算式の間違いやデータの勘違い、アルゴリズムそのものに誤りがある場合など、様々なケースが考えられます。

なまじプログラムが動いてしまうため、致命的でない場合、まったく気付かれないで一般に発表されてしまうこともあります。

ちなみに、明らかに異常動作であっても、作者がそれを承知している場合でプログラムの目的とする作業に支障がないときには、それをバグと呼ばずに仕様としてしまうこともあります。仕様というのは、そのプログラムを作る目的やその実現方法をいうので、この場合、それは異常動作ではなく、始めからそう動作するように作ってあるのだ、という意味になります。

7. お気に召すまで調整を

デバッグが済んで、プログラムが動くようになれば、これでようやくできあがりなのですが、実際に試してみると物足りない部分やバランスの悪さに気付くことがあります。

そういう時には、こころゆくまで手直しをしましょう。

気を付けなければならないのが、自分で作ったプログラムであるため、他人に分かりにくく、難しいゲームとなってしまいがちな点です。作った人間はディスプレイに映し出されるキャラクタがどうしてそういう動きをしているか知っているため、内容や操作の難しさに気付かないことがよくあります。

そういうときは、周りの人間をつかまえて、テストプレイしてもらいましょう。ちなみに、この講座のサンプルプログラムのテストプレイは私の隣に机があるAK君にお願いしました。おかげで、最初のプログラムはゲームとして難しいということが分かりました。

この作業は、プログラムを一般に公表した後も行うことがあります。市販ソフトでいうところのバージョンアップです。

バージョンアップが行われる理由は、実際に使ってみて不具合があった部分の修正や、後になって見つかったバグの対処、人からの要望の実現などといった改造が行われることになります。

8. 完成したら遊ばせて

というわけで、実際にプログラムを作る手順を示しつつ簡単なゲームを作ってみました。

実をいうと、七色紀行さんからは、「BASIC入門をお願いします」と依頼されていたのですが、いつのまにかプログラミング入門になってしまいました。しかし、この講座はこの本のために特別に書いたものです。この本を手にした人は得しましたね(?)。

ともあれ、BASICによるプログラミングは、習うより慣れろで、理論を勉強するよりも、実際にいじって試した方が何倍も勉強になります。理屈は後からついてくるものです。

難しいことは理論屋にまかせて、ポケコンを楽しみましょう。そして、自分が楽しんだらそれを他の人にもわけてあげましょう。そうすれば、自分自身もっと楽しめると思いますよ。

そして、面白いゲームが完成したら、私にも楽しませて下さい。約束ですよ。

1995.04.15 滝本飛沫(たきもとしぶき)
2004.10.23 PDF化