C 言語を知っている人向けの EXCEL VBA の超簡単入門

茨城大学工学部情報工学科
新納 浩幸

shinnou@mx.ibaraki.ac.jp

目 次

第 1 章　プログラムの記述・セーブ・実行　3

第 2 章　基本的な言語の文法　15

2.1　変数の型 . . . 15
2.2　型に対する演算 . . . 15
2.3　配列 . . . 16
2.4　条件式 . . . 16
2.4.1　数値の比較 . . . 16
2.4.2　文字列の比較 . . . 17
2.4.3　論理演算子 . . . 17
2.5　条件分岐 . . . 17
2.6　繰り返し制御 . . . 18
2.7　サブルーチンと関数 . . . 18
2.8　基本関数 . . . 19
2.9　練習 . . . 19

第 3 章　出力の方法（Excel のシートを使え）　21

3.1　セルに数値、文字列を入れる . . . 21
3.2　セルの指定方法 . . . 21
3.3　練習 . . . 24

第 4 章　ワークシート関数の利用　25

4.1　引数が範囲 . . . 25
4.2　出力が範囲 . . . 27
4.3　練習 . . . 28

第 5 章　入力の方法（InputBox の利用）　29

5.1　InputBox の利用 . . . 29
5.2　練習 . . . 30

第 6 章　入力の方法（UserForm の利用）　31

6.1　UserForm の挿入 . . . 31
6.2　UserForm の実行 . . . 32
6.3　Tips . . . 33

はじめに

このテキストはシステム工学科 B コース３年生の主題別ゼミのために書いたものです。
　私の主題別ゼミのテーマは「Excel VBA 入門」です。以下の文章は、このテーマの紹介文として学科に提出したものです。

「Excel は単に表やグラフを作成するだけのソフトではない。Excel は VBA というプログラミング言語を内蔵し、Excel 上のデータに関して様々な処理を行うプログラムを作成することができる。ここでは VBA によるプログラミングの基礎を学ぶ。最終的には Excel の sheet 上で表現された連立１次方程式を解くプログラムを作成する。若干の GUI の部分も付加する。C 言語以外のプログラミング言語に触れることはプログラムに関する知識を深める。またプログラムを単にツールとして使いたいのであれば、 VBA のような簡易かつ強力なプログラミング言語を使うことは有効な選択肢であり、VBA を修得する意義は大きい。」

　要は VBA はおもしろく、ためになるということです。がんばって習得して下さい。
　ゼミはこのテキストに沿って進めます。基本的に各章を１週でやります。ゼミの時に私がざっと説明して、各自練習問題をやって、その時間の最後に章末の課題を解くという流れでやります。
　このゼミが終了するときには、目標である連立１次方程式を解くプログラムは、すぐに作成できる状態になっていると思います。それを作成して終わりです。

第1章　プログラムの記述・セーブ・実行

まず Excel 上でマクロ（プログラム）をどうやって記述して、どうやってセーブして、どうやって実行するかを知らないと話になりません。これはプログラム言語の文法とは別の問題です。

ステップ 1 Excel を起動する。

これは誰でも出来るはず。

ステップ 2 VBE というプログラム開発環境を起動する。

これは図で示すように「ツール」→「マクロ」→「Visual Basic Editor」を選択します。ショートカットで `ALT + F11` でもよい。

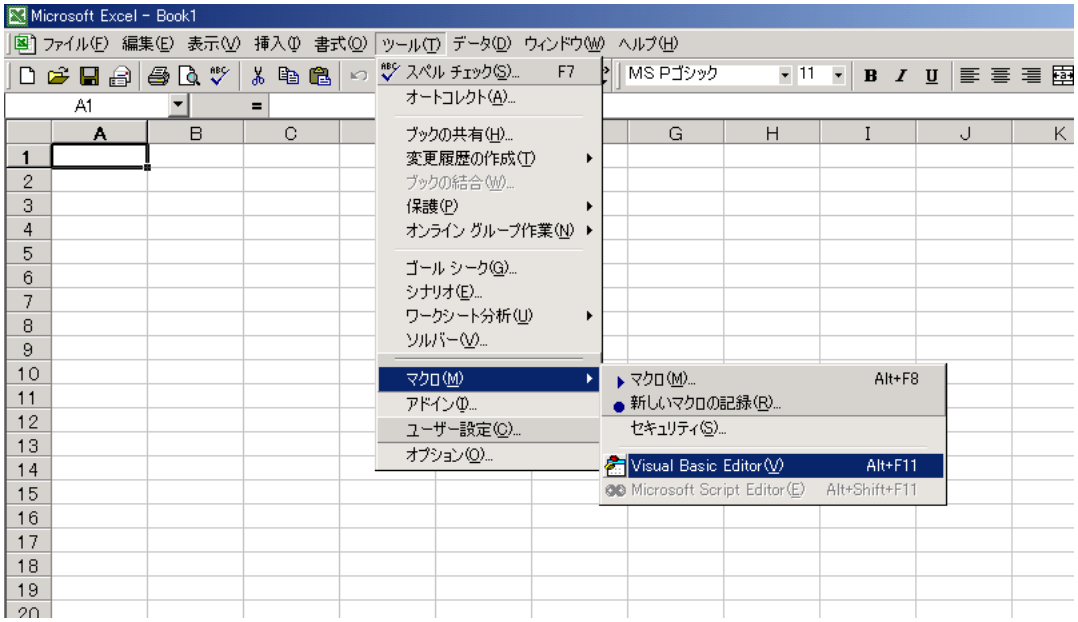

図 1.1: VBE の起動

あーっとビックリ、図のような画面になりました。

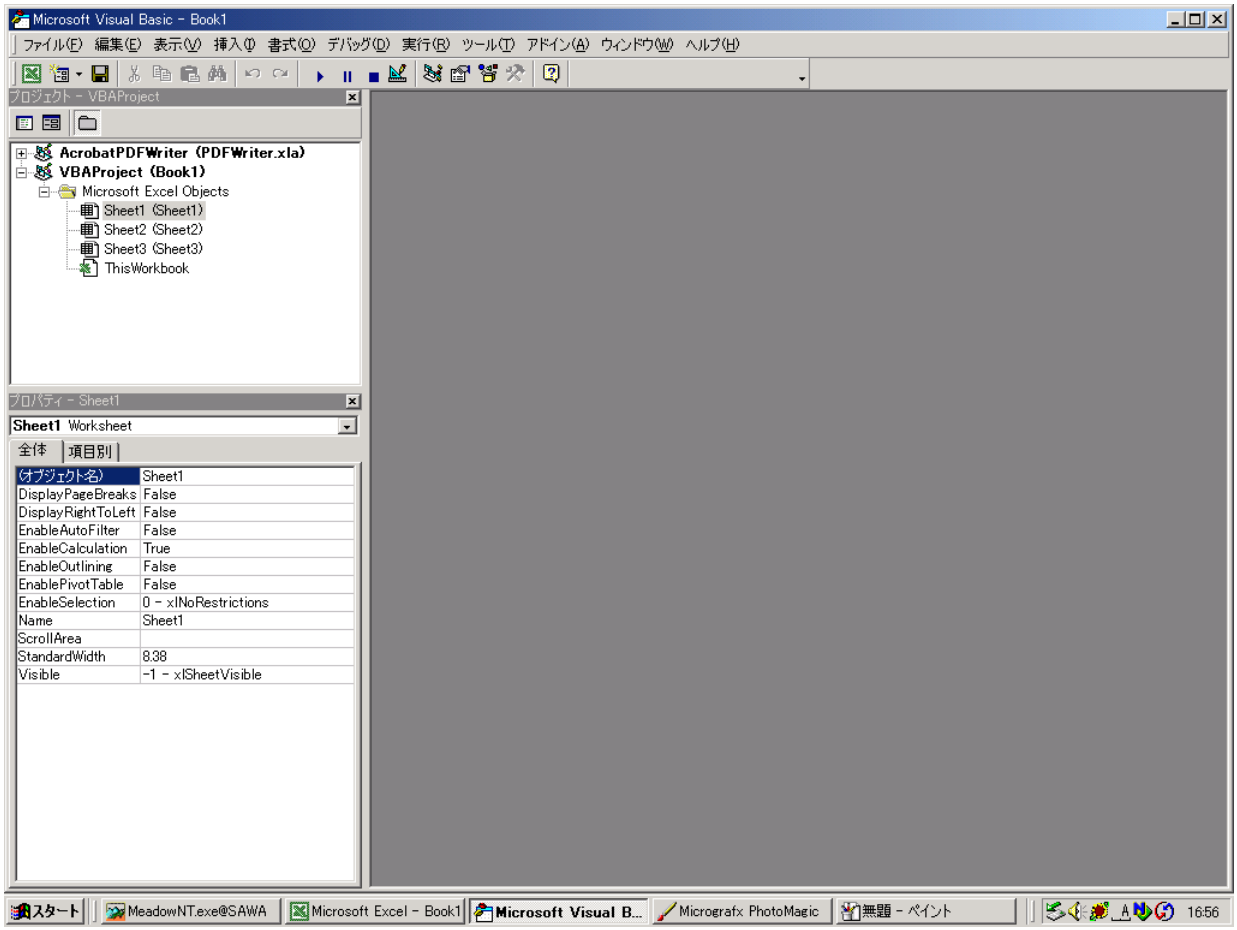

図 1.2: VBE の画面

ステップ 3 モジュールを挿入する。

意味不明でしょうけど、気にしない。とりあえず、「挿入」→「標準モジュール」を選択します。

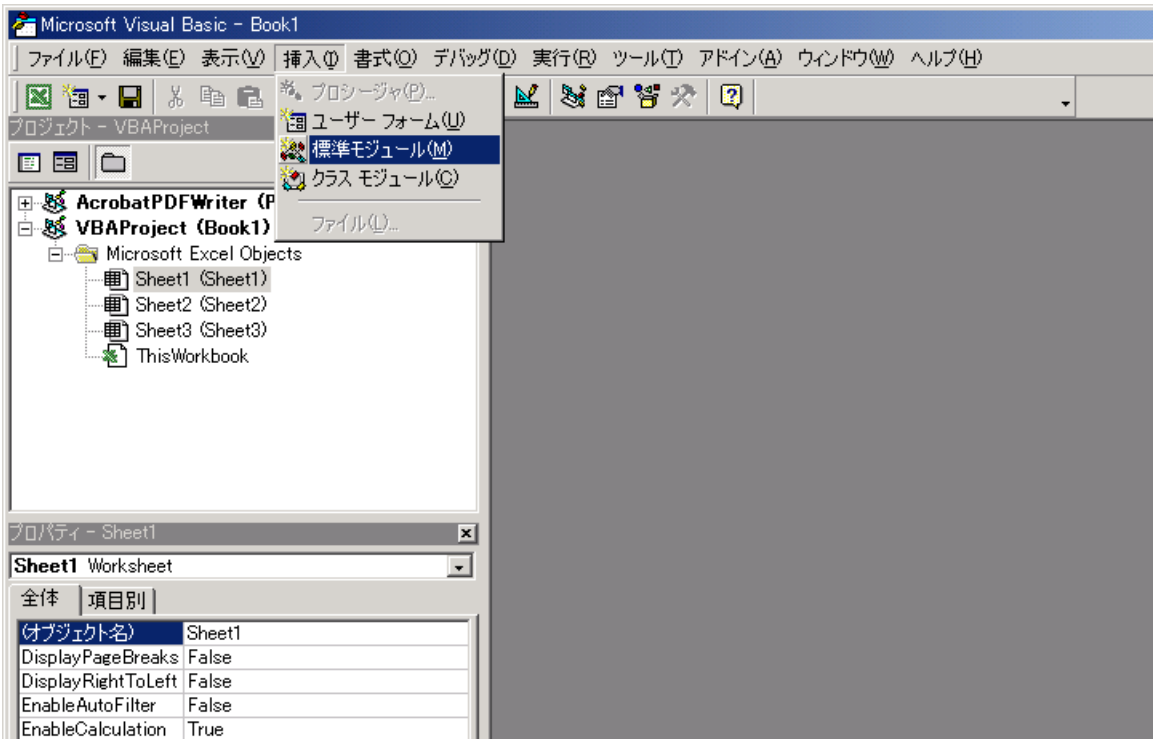

図 1.3: 標準モジュールの選択

右側に Module1 というウインドウが現れます。

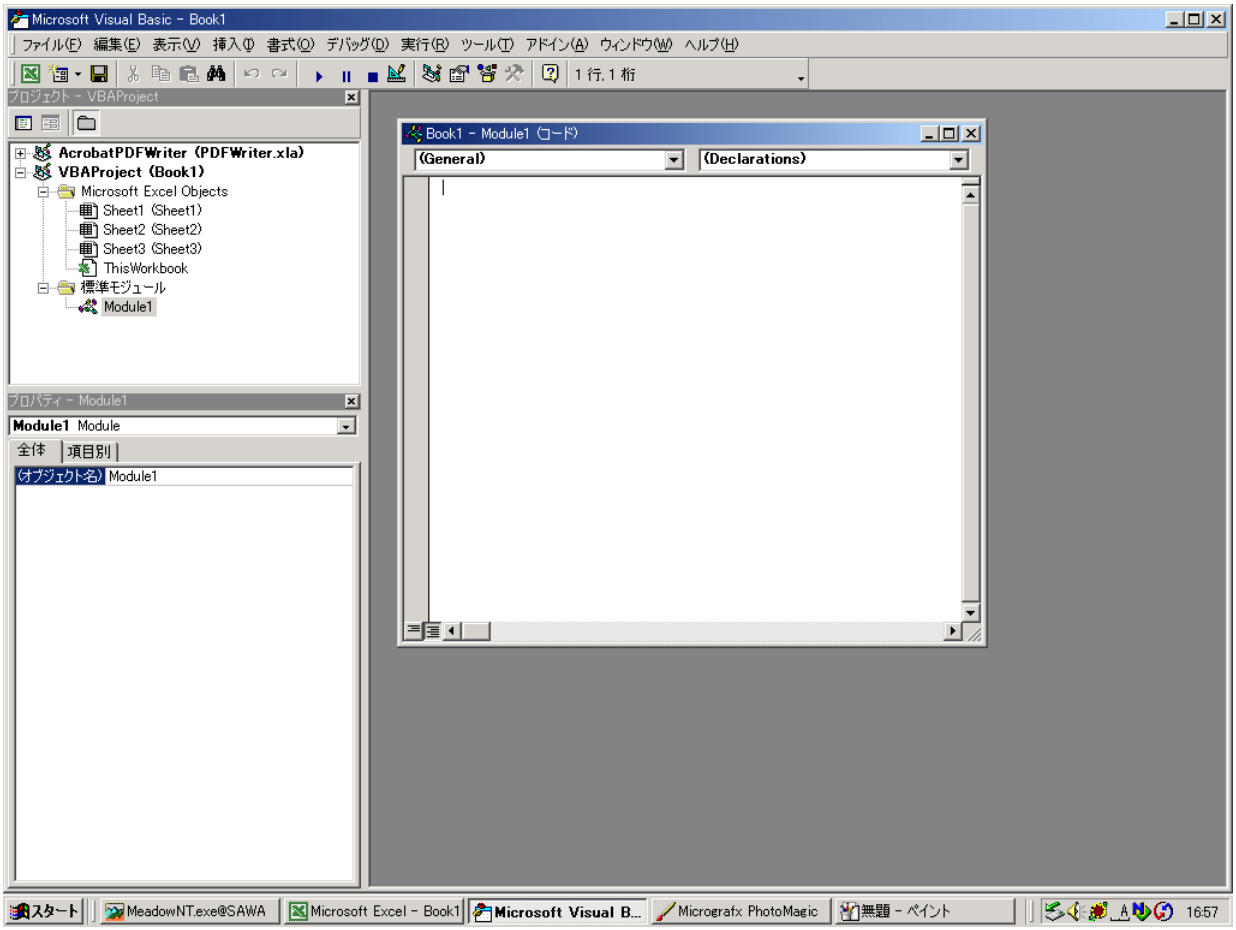

図 1.4: モジュールの出現

ステップ 4 モジュールに名前をつける。

左側の下のウインドウで、「オブジェクト名」と表示されている欄があるはずです。ここに適当な名前を入れます。ここでは `MySample` にしています。

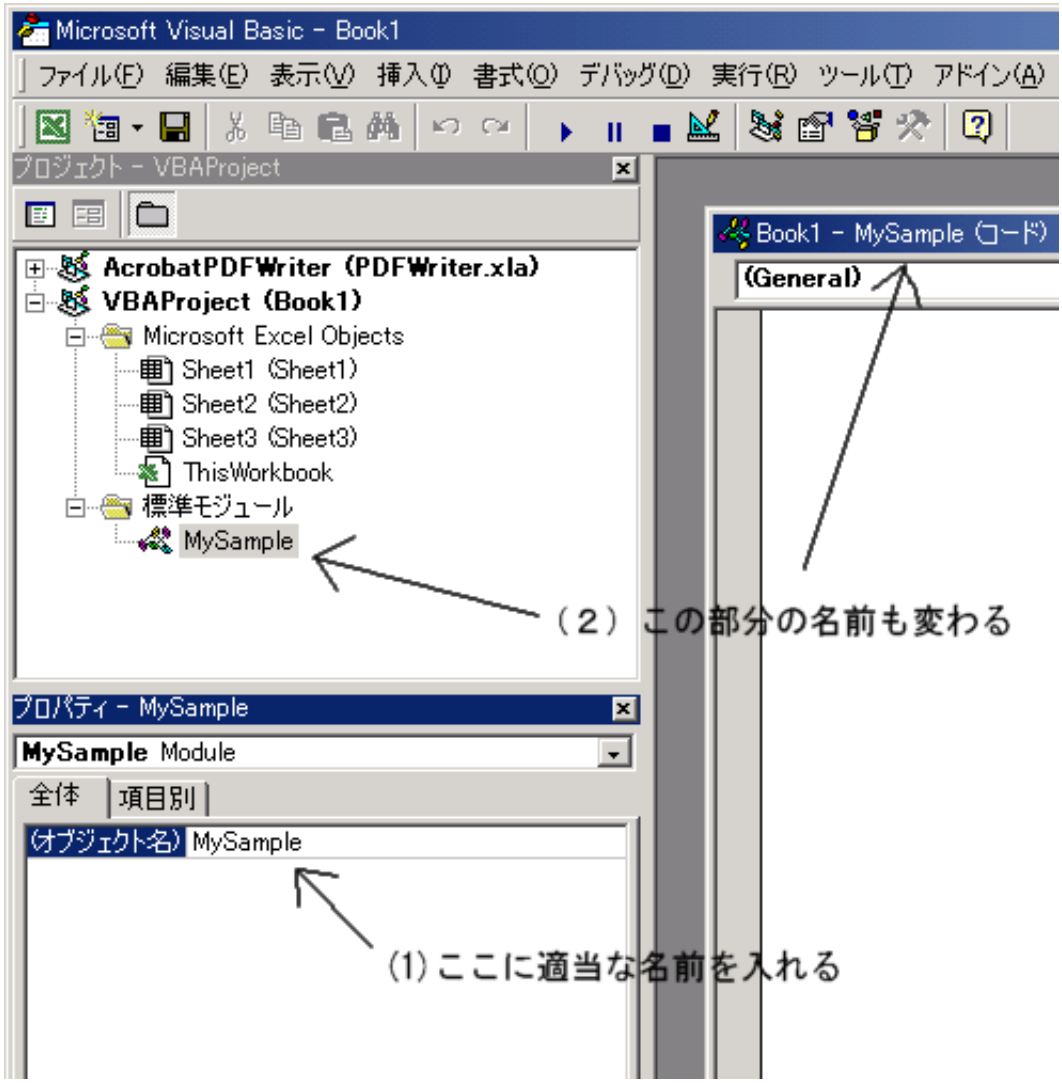

図 1.5: モジュール名の付与

ステップ 5 プログラムを書く。

プログラムは右側のウインドウ（この場合 MySample（コード））に書きます。

ここでは簡単な例として以下のようなプログラムを書いてみます。プログラムの名前は MyTest にしています。書いていると、色々と補完してくれるかも知れません。慣れると便利です。

```
Sub MyTest()
    MsgBox "こんにちは"
End Sub
```

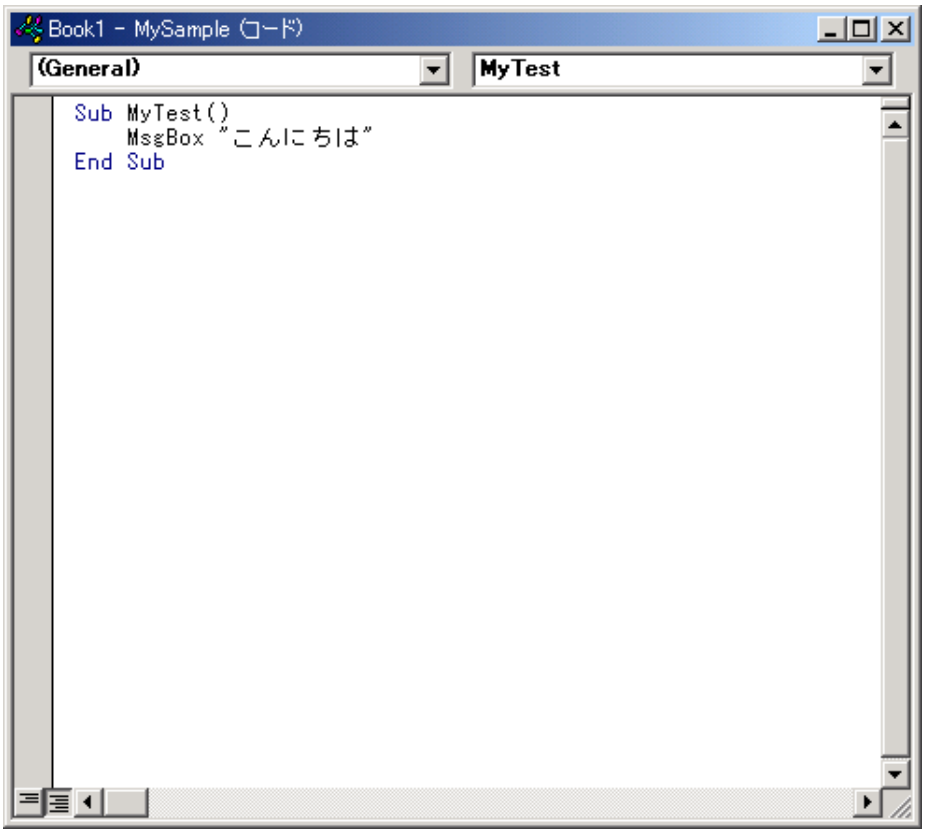

図 1.6: プログラムの記述

ステップ 6 VBE をぬけて Excel の画面に戻る。

「ファイル」→「終了して Microsoft Excel へ戻る」を選択してください。Excel の画面に戻ります。

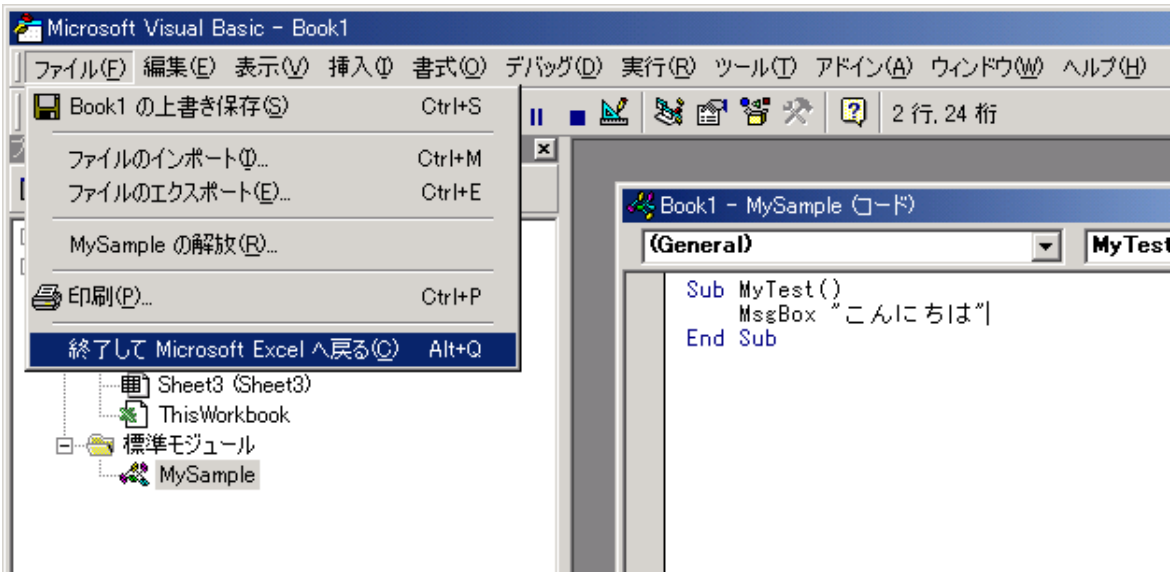

図 1.7: VBE から Excel に戻る

ステップ 7 プログラムを実行する。

先のプログラムを動かしてみます。「ツール」→「マクロ」→「マクロ」を選択します。ショートカットで `ALT + F8` でもよいです。

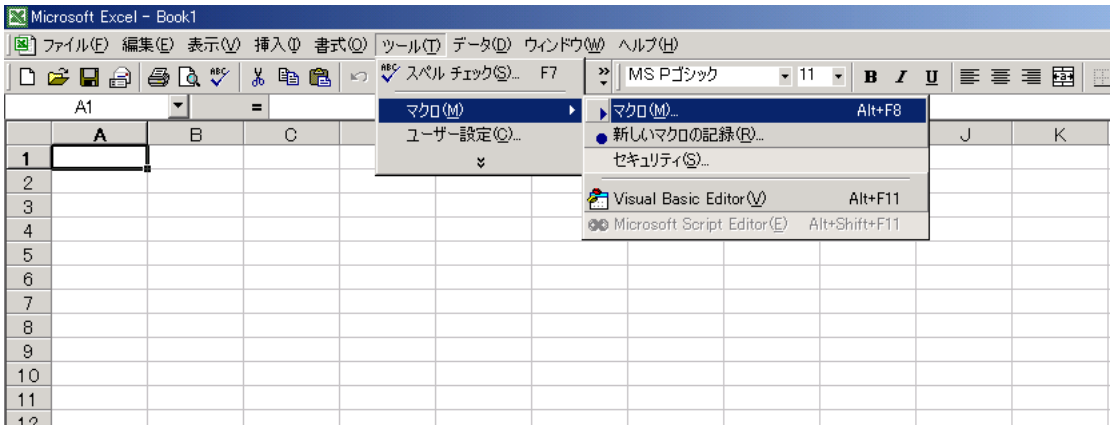

図 1.8: プログラムの実行 (1)

利用できるプログラムの一覧が表示されたウインドウが現れます。そこで先の MyTest を選んで実行してください。

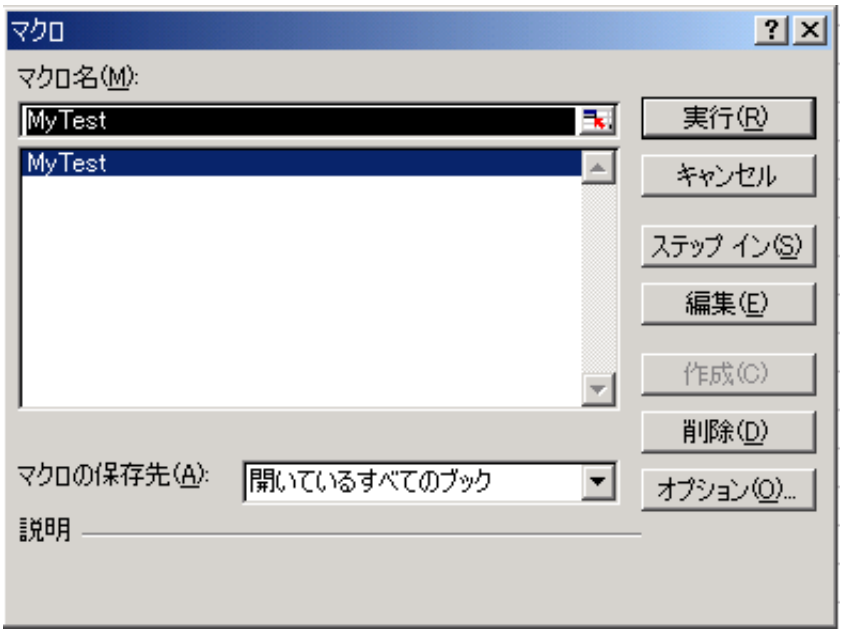

図 1.9: プログラムの実行 (2)

このプログラムの結果は以下のようなウインドウが表示されるだけです。

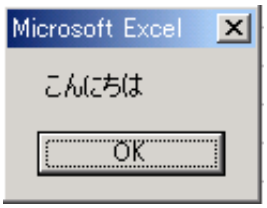

図 1.10: プログラムの実行結果

ステップ 8 プログラムを保存する。

ステップ 2 でやったように VBE の環境に移動してください（ `ALT + F11` です)。

そこで右側の MySample(コード）のウインドウを一度アクティブにしてから、「ファイル」→「ファイルのエクスポート」を選択してください[1]。

[1] VBA ではプログラムを保存することを「エクスポート」と言います。

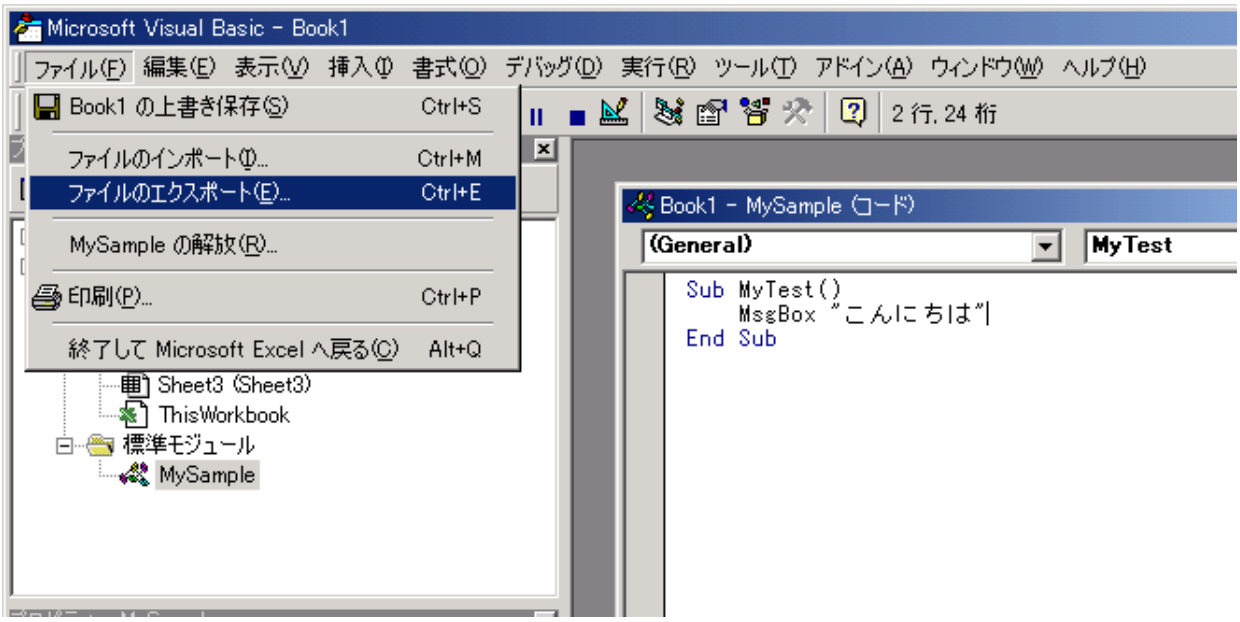

図 1.11: プログラムの保存

ファイルダイアログのウインドウが出ますので、適当なディレクトリに保存してください。デフォルトのファイル名はステップ４でつけたモジュールの名前になっています。ここではこのファイル名のまま保存してください。

保存すると、MySample.bas というファイルが出来ています。これはテキストファイルです。中身を見ると以下のようになっています。

```
Attribute VB_Name = "MySample"
Sub MyTest()
    MsgBox "こんにちは"
End Sub
```

１行目はモジュール名が書かれていますが、その他はステップ５で書いた内容が入ったファイルです。VBE 上のエディタが不満な人は、好みのエディタでこのファイルを直接作成してもかまいません。ただ実行するためには Excel を起動しないとダメです。私は大まかに自分のエディタでプログラムを書いて、こまかい修正だけ VBE でやっています。

次のステップで Excel を終了しますが、このままだとこのプログラムが Excel のファイルにくっついた形になります。Excel のファイルとプログラムを分離するために、プログラムを解放します。MySample(コード) のウインドウを一度アクティブにしてから、「ファイル」→「MySample の解放」を選んでください。

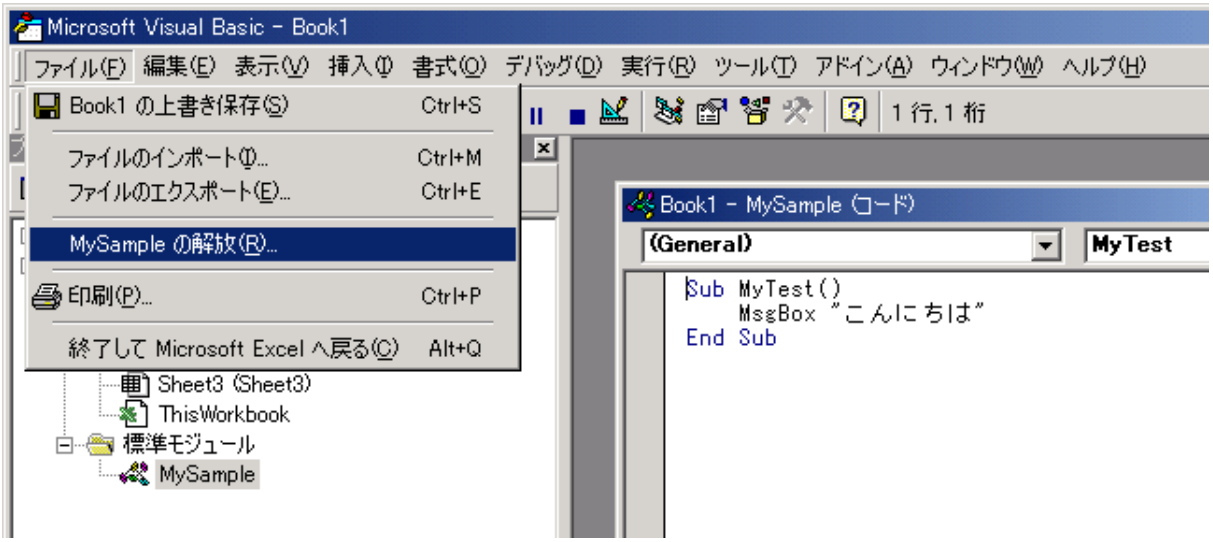

図 1.12: プログラムの解放

「削除するまえに MySample をエクスポートしますか」と聞かれますが、さっき保存しておいたので、ここでは保存しないくてかまいません。「いいえ」を選択します。でも、もちろん、保存してもいいです。気になる人は「はい」を選択して保存して下さい。

ステップ 9 Excel を終了する。

ステップ 6 で行ったように Excel の画面に戻って終了すればよいです。

ステップ 10 プログラムをインポートする。

新たに Excel を立ち上げて、そこで先ほどのプログラムを動かしたいとします。
そんなときは、プログラムをインポートします。
まずステップ 2 でやったように VBE に移動して、そこで「ファイル」→「ファイルのインポート」を選んでください。あとはフィルダイアログのウインドウが出ますので、先ほど保存しておいた MySample.bas を選択すればよいです。

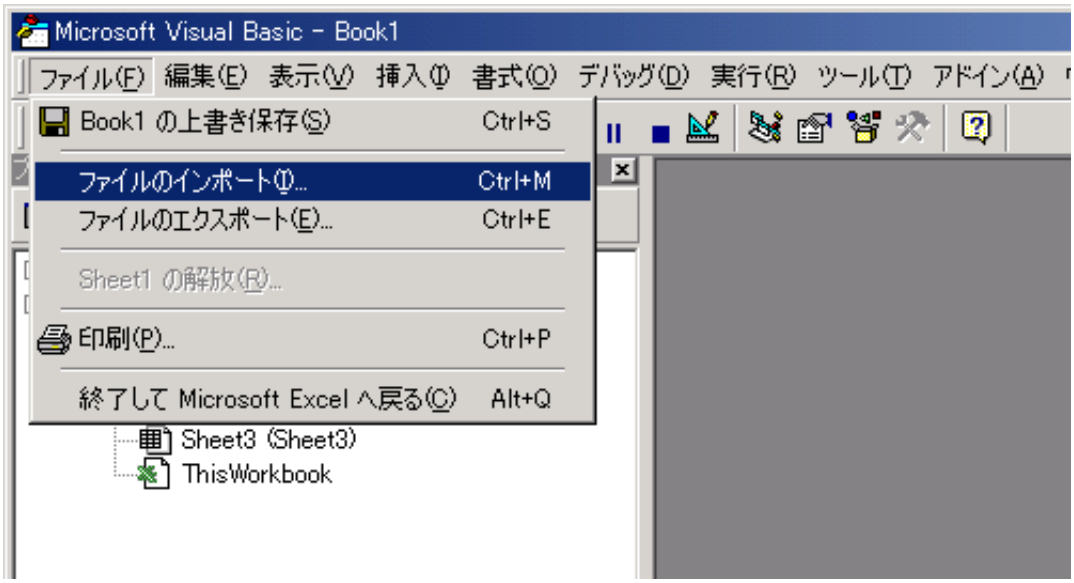

図 1.13: プログラムのインポート

インポートされると、先ほどと同様、左上のプロジェクトウインドウの中の標準モジュールの中に MySample が追加されていることが確認できます。

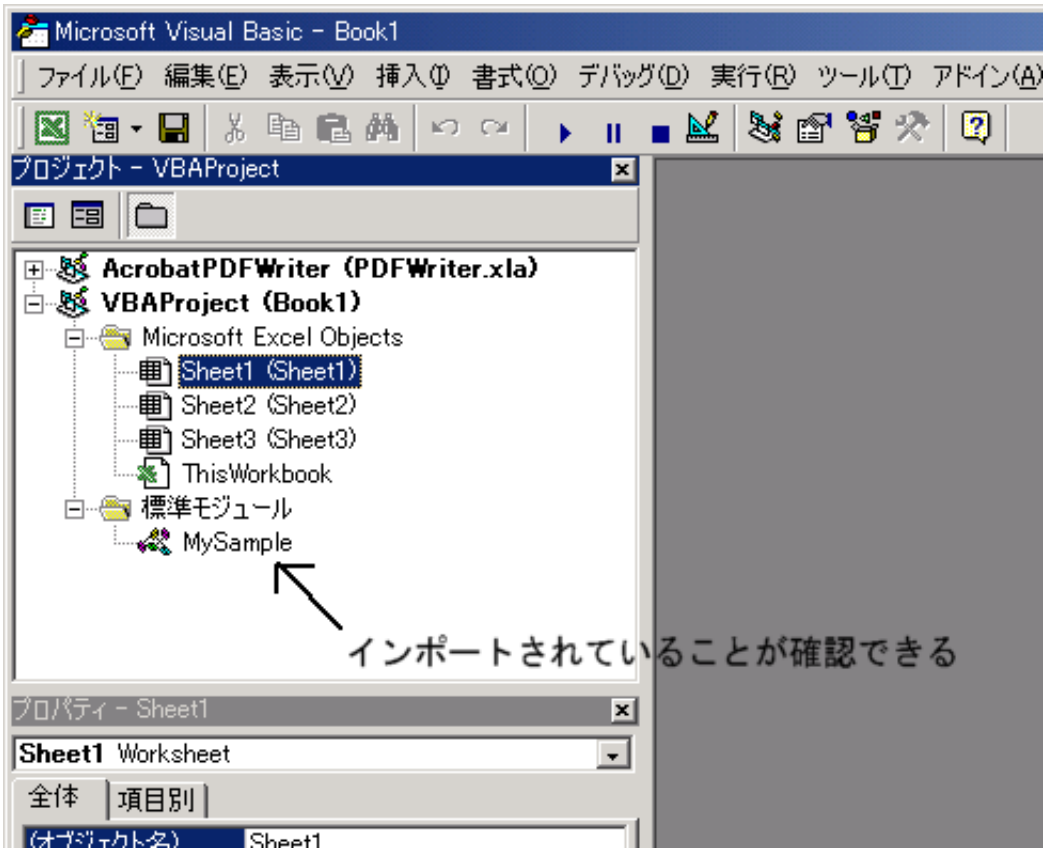

図 1.14: インポートされた結果

第2章　基本的な言語の文法

C 言語と同様、VBA も文の並びになっています。

ただ C 言語の場合、１行の中に複数の文があってもよいし、複数行にわたって１文を書くこともできます。ところが VBA は１行１文です。これが大きな違いです。もしも長い文の場合はアンダーバー _ の後に改行をいれて、複数行にわたって記述できます。

１行１文だとすると、複数の文をまとめて１文にするような場合はどうするのでしょうか？ 基本的には開始と終了に対応する文があります。

C 言語に慣れていると、非常に冗長な記述に見えますが、それが VBA なのでしかたないです。

2.1　変数の型

VBA で利用できる変数の型として以下のものがあります。他にもありますが、とりあえず、以下のものだけ覚えておけば良いでしょう。型の宣言は、以下の形で行います。

`Dim`　変数名　`As`　型

- 整数型　`Dim i As Interger`

  -32768 から 32767 までの整数

- 長整数型　`Dim k As Long`

  -2147483648 から 2147483647 までの整数

- 倍精度実数型　`Dim dr As Double`

  倍精度の実数

- 単精度実数型　`Dim fr As Single`

  単精度の実数

- 文字列型　`Dim str As String`

  文字列

- Boolean 型　`Dim flg As Boolean`

  True あるいは False。値はこの２つのみ。

2.2　型に対する演算

（長）整数型、（単 or 倍精度）実数型に対する演算の基本は以下のものです。

```
+     足し算
-     引き算
*     掛け算
/     割り算
```

少し変わったものとしては以下があります。

```
\      整数割り算    10\3 --> 3
^      べき乗        10^3 --> 1000
mod    余り          10 mod 3 --> 1
```

文字列に対する演算の基本は以下です。

```
&      文字列の連結    "abcd" & "efg" --> "abcdefg"
```

2.3 配列

例えば、以下のように宣言します。

```
Dim ary(100) As Double
```

これは Double 型の変数 `ary(0)` から `ary(100)` までの 101 個 宣言したのと同じです。要素数が C 言語とは違うので注意して下さい。

また多次元配列も使えます。例えば、以下は 3 行 11 列の Double 型の配列です。

```
Dim ary(2,10) As Double
```

2.4 条件式

条件分岐や繰り返しの処理の中で必要となる条件式の形は、数値の比較、文字列の比較、Boolean 型を返す関数、そして True あるいは False です。

2.4.1 数値の比較

=	等しければ真
<>	等しくなれれば真
<	左辺が右辺よりも小さければ真
<=	左辺が右辺以下なら真

表 2.1: 数値の比較

2.4.2 文字列の比較

=	等しければ真
<>	等しくなれれば真

表 2.2: 文字列の比較

等しいかどうかの比較も代入も = を使うので、C 言語とは違います。注意して下さい。

2.4.3 論理演算子

Not	否定 Not (A > B)
And	論理積 (A >B) And (C < D)
Or	論理和 (A >B) Or (C < D)

表 2.3: 論理演算子

2.5 条件分岐

```
IF（条件式） Then

   処理

End If
```

```
IF（条件式） Then

   処理 1

Else

   処理 2

End If
```

2.6 繰り返し制御

```
Do While (条件式)

   処理 1

Loop
```

```
For カウンタ変数 = 初期値 To 終了値 <Step ステップ値>

   処理

Next カウンタ変数
```

2.7 サブルーチンと関数

返す値があるものが関数で、ないものがサブルーチンと呼ばれています。C 言語ではどちらも関数です。

サブルーチンは以下の形で書きます。

```
sub  サブルーチン名 (引数, 引数, ... , 引数)

   本体

End Sub
```

サブルーチンを強制的に抜けるには Exit Sub です。
以下は例です。引数の型の書き方に注意してください。

```
sub tasizan (A as Integer, B as Integer)
    MsgBox  A+B
End Sub
```

呼び出すときは、以下の形です。

```
call tasizan(3,4)
```

関数は以下の形で書きます。

```
Function 関数名（引数，引数，... ，引数） As 返り値の型

   本体

End Function
```

返り値は 関数名 = ○○ のように、関数名に代入します。関数を強制的に抜けるには Exit Function とします。

以下は例です。ただし無理やりなので不自然なプログラムです。

```
Function dif (A as Integer, B as Integer) As Integer
    if (A > B) Then
       dif = A - B
       Exit Function
    End If
    dif = B - A
End Function
```

関数を呼び出すときは以下の形です。結果をある変数（下記の場合 result）で受けないと、意味がありません。

```
result =  dif(3,2)
```

2.8 基本関数

文字列に対する基本関数は省略します。数学関係の基本関数は Excel 内の関数を利用するので、これもとりあえずここではパスです。

2.9 練習

問題 1 10000 以下の数値で最も大きな素数を求め表示せよ。

引数で与えられた正の整数 n が素数かどうかを判定する Prime という関数を作ります。Prime は n が素数であれば True を返し、素数でなければ False を返すという返り値が Boolean 型の関数です。Prime の中身は簡単に、2 から n-1 まで順番に n を割ってゆき途中で割りきれる数があれば、素数でないと判断し、最後まで割りきれなければ素数と判断します。

最後に答の表示ですが、まだ習っていないので無理です。とりあえず、

```
  MsgBox  文字列
```

は文字列を表示する関数として利用することにします。
解答は例えば以下のような感じです。

```
Sub mondai1()
    Dim i As Integer
    For i = 10000 To 2 Step -1
        If (Prime(i)) Then
            MsgBox "答えは" & i & "です"
            Exit Sub
        End If
    Next i
End Sub

Function Prime(n As Integer) As Boolean
    Dim i As Integer
    For i = 2 To n - 1
        If ((n Mod i) = 0) Then
            Prime = False
            Exit Function
        End If
    Next i
    Prime = True
End Function
```

問題 2 最大公約数を求める関数を作成し、39597 と 100667 の最大公約数を求め、その結果を表示せよ。

最大公約数の求め方は、小さい方の数を 1 引きながら、その数で、両方の数値を割ってみるという戦略でやりましょう。
どちらの数も大きいので Long でないとダメです。

第3章　出力の方法（Excel のシートを使え）

プログラムで最初に習うべきもっとも大事な処理は、「計算結果をどうやって表示するか」です。C 言語で言うとことの　printf("Hello World!") です。

VBA の場合、計算結果を表示する方法はいくつかあります。最も基本的な方法は、前章で使った　MsgBox を使う方法です。

MsgBox は数個の結果を文字列として表示するには簡単です。しかし、実際の Excel 上で行う計算は大きな表を扱うことが多いと思います。その場合、求めるべき計算結果も大きな表になるでしょう。その場合、　MsgBox で表示するのは無理です。

結論を言えば、Excel の Sheet に計算結果を書き出すのが現実的です。そうすれば、カットアンドペーストで計算結果を再利用することも可能です。

3.1　セルに数値、文字列を入れる

Sheet に計算結果を書き出す基本は、指定の Sheet 上のあるセルに数値や文字列を入れることです。例えば、以下の例では Sheet1 のセル B3 に 12.5 という数値を入れています。

```
Worksheets("Sheet1").Range("B3") = 12.5
```

Worksheets("Sheet1") の部分が省略されると、現在、アクティブになっている Sheet が対象になります。”Sheet2” をアクティブにするには、以下のようにします。

```
WorkSheets("Sheet2").Activate
```

3.2　セルの指定方法

シート上のあるセルを指定する方法がいくつかあります。基本的には Excel で通常使っている A1 形式と呼ばれている B3 のような表現方法を使います。でも、これだと、「B3 から右に 3 つ行って、下に 1 つ下がった位置のセル」などいうセルを指定したいときに、ちょっと面倒です。

プログラムでセルの位置を指定するのは、インデックス番号を利用する方が便利です。これはセルの行と列のインデックス番号を利用してセルの位置を指定する方法です。B3 の場合、行のインデックスは 3 で列のインデックスは 2 なので、Sheet1 のセル B3 に 12.5 という数値を入れるには以下のようにします。

```
Worksheets("Sheet1").Cells(3,2) = 12.5
```

問題 3 10000 以下の素数をすべて求めシートに出力せよ。ただし、1 行に 10 個数値を出力したら、次の行に移ること。

出力位置の行を i、列を j としておきます。j が 10 を超えたら、 `j = 1` 、 `i = i+1` にします。

```
Sub mondai2()
    Dim n As Integer
    Dim i As Integer
    Dim j As Integer
    i = 1
    j = 1
    For n = 2 To 10000 Step 1
        If (Prime(n)) Then
            Cells(i, j) = n
            If (j = 10) Then
               j = 1
               i = i + 1
            Else
               j = j + 1
            End If
        End If
    Next n
End Sub

Function Prime(n As Integer) As Boolean

    ここはさっきと同じなので、省略します

End Function
```

セルに数値を入れることができるのならば、セルに入っている数値を取り出すことも可能です。以下は Sheet1 のセル B3 に入っている数値を取り出して、変数 i に入れています。

```
i = Worksheets("Sheet1").Cells(3,2)
```

ここで注意が必要です。Excel の場合、セル B3 に入っているのは、数値だけとは限りません。文字列もありえますし、数式かもしれません。ですので、変数 i の型を何に設定しているかが問題です。

しかし、結論を言えば、気にしなくてもよいです。整数値が入っていると仮定しているなら、整数型で受ければよいし、実数値が入っていると仮定しているなら、実数型で受ければよいです。また数式の場合は数式を受ける形式で書けばよいです。もしも仮定している型でないデータであったら、、、、こういう例外的な処理を入れるかどうかが問題です。個人が使うプログラムでは、あまりそこらに力を入れなくてもよいと思います。

問題 4 A1:E5 と G1:K5 のそれぞれに 5 × 5 の行列が入っているとする。A1:E5 に入っている行列を A、G1:K5 入っている行列を B とする。行列の積 A × B の行列を A7:E11 に書き出せ。

```
Sub mondai3()
    Dim i As Integer
    Dim j As Integer
    Dim k As Integer
    Dim sum As Integer
    Dim ai As Integer
    Dim aj As Integer
    Dim bi As Integer
    Dim bj As Integer
    Dim ci As Integer
    Dim cj As Integer

    ai = 1   ' 行列 A の左上の行
    aj = 1   ' 行列 A の左上の列
    bi = 1   ' 行列 B の左上の行
    bj = 7   ' 行列 B の左上の列
    ci = 7   ' 結果 AB の左上の行
    cj = 1   ' 結果 AB の左上の行

    For i = 0 To 4 Step 1
       For j = 0 To 4 Step 1
       sum = 0
          For k = 0 To 4 Step 1
             sum = sum + Cells(ai + i, aj + k) * Cells(bi + k, bj + j)
          Next k
          Cells(ci + i, cj + j) = sum
       Next j
    Next i
End Sub
```

3.3 練習

問題 5 A1:A10 の範囲に入っている 10 個の数値群と B1:B10 に入っている 10 個の数値群との共分散を求めよ。

共分散の定義は以下です。

$$\frac{1}{n}\sum_{i=1}^{n} x_i y_i - \left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)\left(\frac{1}{n}\sum_{i=1}^{n} y_i\right)$$

参考までに、Excel では共分散を求める関数があります。それは `covar(A,B)` です。

第4章　ワークシート関数の利用

実はここまでの話だと、VBA というプログラム言語を使う理由は全くありません。他の言語を利用した方がよっぽど簡単にプログラムが書けます。

VBA を使うメリットの 1 つはワークシート関数が利用できることです。例えば、C 言語で逆行列を求めるプログラムを書くのは、結構大変です。ところが VBA ではそこらの数学、統計関係の関数はワークシート関数としてそろっているので、単に関数を呼び出すだけで出来ます。

どんな関数があるか一覧を出すのは大変なので、必要そうなものだけを使い方と一緒に解説します。

4.1　引数が範囲

ワークシート関数は多くの場合、引数がセルの範囲になっています。

例えば、関数 max は指定した範囲の中の各セルに入っている数値の中で、最大値を返す関数です。これを使ってみましょう。

```
Sub mondai3()
    Dim m As Integer
    m = Application.WorksheetFunction.Max(Range("A2:D5"))
    MsgBox m
End Sub
```

上の例では、A2:D5 の範囲を　`Range("A2:D5")` として、これを関数 Max の引数にしています。関数 Max の前に

```
Application.WorksheetFunction.
```

がついています。これは VBA がオブジェクト指向の言語であることの片鱗を示しています。この解説は長くなるので、省略です。そういうものだと思っていてください。ただ、ワークシート関数を使うたびに、こんな長たらしい文字列を書くのは大変なので、ちょっとテクニックがあります。

```
Sub mondai3()
    Dim m As Integer
    Set WF = Application.WorksheetFunction
    m = WF.Max(Range("A2:D5"))
    MsgBox m
End Sub
```

上記のように、最初に Application.WorksheetFunction のオブジェクトに名前をつけておけばいいのです。名前を付けるには、上記のように Set という命令を使います。

```
Set 変数 =  オブジェクト
```

こうすると、以降そのサブルーチン内では、Application.WorksheetFunction の部分をそのインスタンス名に置き換えられます。

また範囲の指定の方法として、セルの場合同様、A1 形式を用いるのはプログラムしずらいので、インデックスの形式を使います。 `Range("A2:D5")` は以下のように書きます。

```
Range(Cells(2,1),Cells(5,4))
```

同じ範囲を何度も使う場合には、その範囲に名前をつけておくと便利です。範囲もオブジェクトなので、先ほど使った Set を利用します。

```
Sub mondai4()
    Dim m As Integer
    Set WF = Application.WorksheetFunction
    Set R = Range(Cells(2,1),Cells(5,4))
    m = WF.Max(R)
    MsgBox "最大値は " & m
    m = WF.Min(R)
    MsgBox "最小値は " & m
End Sub
```

もう１つ簡単な例題をやっておきます。前の章で行った問題 5 をワークシート関数を使って解いてみます。

問題 5 A1:A10 の範囲に入っている 10 個の数値群と B1:B10 に入っている 10 個の数値群との共分散を求めよ。

```
Sub mondai5()
    Set A = Range("A1:A10")
    Set B = Range("B1:B10")
    Set WF = Application.WorksheetFunction
    MsgBox "求める共分散は " & WF.COVAR(A,B) & " です"
End Sub
```

4.2 出力が範囲

例えば、行列の積は MMULT という関数を使います。しかし行列の積の結果は行列になり、この結果をシートに出力するにはどうしたらよいのでしょうか。

これは 1 つのテクニックですので、こうするものだと思っていてください。例えば、問題 4 をもう一度解いてみます。

問題 4 A1:E5 と G1:K5 のそれぞれに 5×5 の行列が入っているとする。A1:E5 に入っている方を行列 A、G1:K5 入っている方を行列 B とする。行列の積 A × B の行列を A7:E11 に書き出せ。

行列の積の結果は 5×5 の行列になり、その結果を A7:E11 の範囲に出力したいとします。この場合、以下のように行います。

```
Sub mondai4-2()
    Set AB = Range("A7:E11")
    AB.FormulaArray = "=MMULT(A1:E5,G1:K5)"
End Sub
```

上記の場合、Application.WorksheetFunction の指定は必要なくなります。MMULT の引数になっているのは文字列であることに注意してください。プログラムの中で、A1:E5 などという文字列を作成するのは面倒です。以下のような形も 1 つのテクニックです。

問題 6 A1:E5 と G1:K5 のそれぞれに 5×5 の行列が入っているとする。変数 d に 1 から 5 までの数を適当に入れて、上記 2 つの行列を $d \times d$ の行列と見て、その積を行列の積 A × B の行列を A7 が左上になるように書き出せ。

```
Sub mondai6()
    Dim d as  Integer
    d = 3
    Set A = Range(Cells(1,1),Cells(d,d))
    Set B = Range(Cells(1,7),Cells(d,6 + d))
    Set AB = Range(Cells(7,1),Cells(6 + d,d))
    AB.FormulaArray = "=MMULT(" & A.Address & "," &  B.Address & ")"
End Sub
```

4.3 練習

では、練習をやってみましょう。

問題 7 A1:E5 に 5×5 の行列が入っているとする。G1:K5 にその逆行列を書き出せ。

逆行列を求める関数は `MINVERSE(A)` です。引数の `A` は正方行列に対応する Range になります。

第5章　入力の方法（InputBox の利用）

ここまでで実は自分が必要なプログラムは書けるはずです。でも何か使い勝手が悪いはずです。それはユーザからの入力を受け付けていないからです。

例えば、ここまでのプログラムでは、あるセルの範囲内の数値の平均を求めるプログラムを書く場合、その範囲は予め、プログラムに与えていました。範囲が変わるごとにプログラムのその部分を書き換えるなんて面倒でやってられません。

ここではその範囲をプログラムから尋ねるようにし、その返事からプログラムは処理すべき範囲を決めることにします。

ここまでに学習した話だけで解決するなら、シートのあるセルに入力を書くことにしておき、プログラムはそのセルからデータを読めばいいですが、これはちょっとかっこ悪いです。

別の方法としては InputBox という命令を使うことあげられます。

5.1　InputBox の利用

問題 8 ユーザーからセルの位置を２つ聞く。最初のセルの位置はある範囲 A の左上のセル、２番目に聞くセルの位置は範囲 A の右下のセルとする。範囲 A の中の数値の合計を求め、表示せよ。

範囲 A が求まれば、合計は sum(A) で求まります。どうやって範囲 A を作るかが問題です。

```
Sub mondai8()
    Dim ai As String
    Dim aj As String
    Dim S As Long
    Set WF = Application.WorksheetFunction
    ai = Application.InputBox("左上のセルの位置を入れよ")
    aj = Application.InputBox("右下のセルの位置を入れよ")
    S = WF.Sum(Range(ai + ":" + aj))
    MsgBox "合計は " & S
End Sub
```

Application.InputBox("左上のセルの位置を入れよ") が実行されると、以下のようなウィンドウが表示されます。

図 5.1: InputBox

このウィンドウのテキストボックスに文字列を入れて、 `OK` ボタンを押すと、このウィンドウは閉じて、入力された文字列が `Application.InputBox("左上のセルの位置を入れよ")` の結果としてプログラムに返ります。

5.2 練習

問題 9 ある範囲に入っている正方行列の

- 左上のセルの位置
- 何次の正方行列か

を InputBox を利用して、ユーザから指定させる。次に、その正方行列の逆行列を求め、ユーザの指定したセルの位置に出力させる。出力位置も InputBox を利用して、ユーザから

- 出力先の左上のセルの位置

を指定させよ。

第6章　入力の方法（UserForm の利用）

入力に InputBox を使うのは、それほど悪くはないのですが、プログラムに与えるデータが複数個になっていたり、数種のものからの選択になっているような場合に、InputBox だけで対応するのは無理です。

結論的は、UserForm を使います。VB がこれほど広まっている背景には、ウインドウベースのインターフェイスを持つプログラムが容易にかける事にあります。VB の入門書を読めば、すぐに以下のような入力画面を作ることが出来るでしょう。

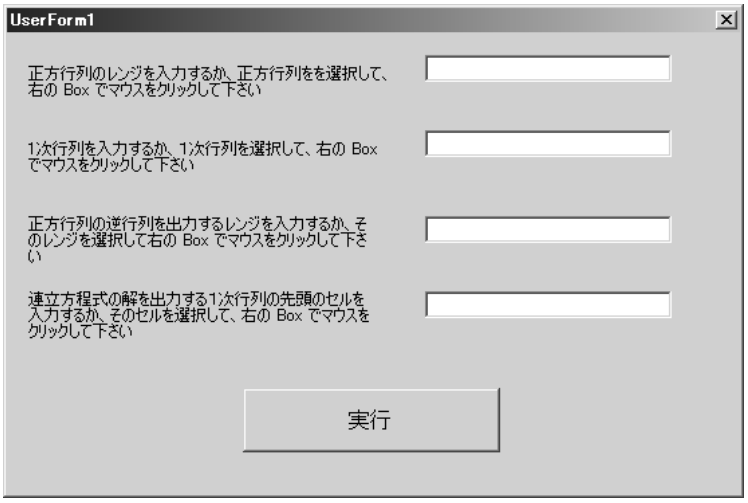

図 6.1: UserForm の例

ここではこのような画面の作り方は省略します。VB の入門書を読んでみてください。大雑把に言えば、コントロールと呼ばれる部品をウィンドウに配置して、そのコントロールのプロパティを変化させて、見た目を好きなように変更します。プログラムとしては、コントロールに与えるイベントでどのような処理を起こしたいかを書きます。簡単です。

ここでは UserForm の利用の概要と Tips をまとめます。

6.1　UserForm の挿入

UserForm を挿入するには、VBE に移り、[挿入] →「ユーザーフォーム」を選択します。以下のような画面になるはずです。

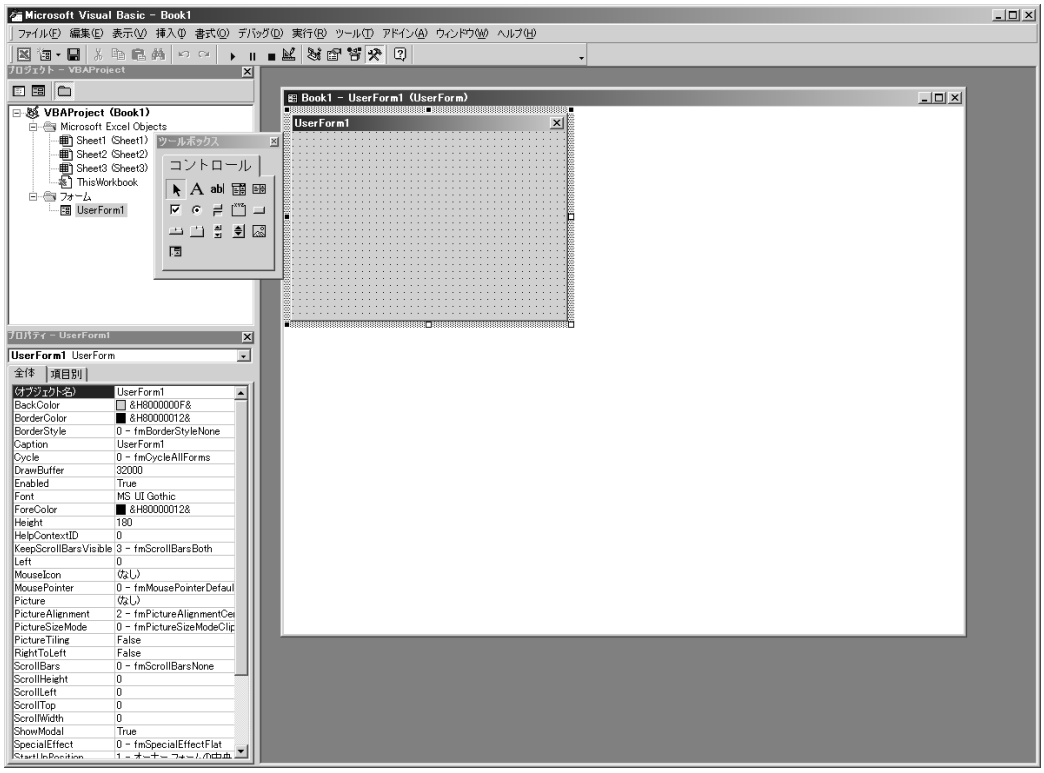

図 6.2: UserForm の挿入

ここからコントロールを配置して、プログラムを書けば良いです。このやり方は、VB と全く同じです。ちまたにある VB の入門書を読めば分かると思います。またこの部分のプログラム内には、これまでに学んだこと（例えば、WorksheetFunction など）をそのまま使えます。

UserForm のエクスポートやインポートは標準モジュールでの方法と同じです。

6.2 UserForm の実行

UserForm は VBE の環境内では、ツールバー内にある「進むボタン」を選択することで実行できますが、Excel のシートの画面内では実行するのが面倒です。このため、マクロから UserForm を実行するようにします。

通常、実際の処理も UserForm 内で行うでしょうから、マクロは以下の 1 行を入れるだけになります。

```
Sub UFstart()
    UserForm1.Show
End Sub
```

「UserForm1」はオブジェクト名です。変更していなければ、「UserForm1」になっているはずです。

6.3 Tips

以下は Excel の VBA で、UserForm を使う際の Tips です。

● UserForm から Excel のシートの方へマウスを動かせるようにする

UserForm では、ユーザがある範囲を指定して、そのレンジをある変数に設定する、といった処理を行いたい場合が多いと思います。この場合、UserForm のプロパティの「ShowModal」を「False」に設定します。default は 「True」なので、変更が必要です。

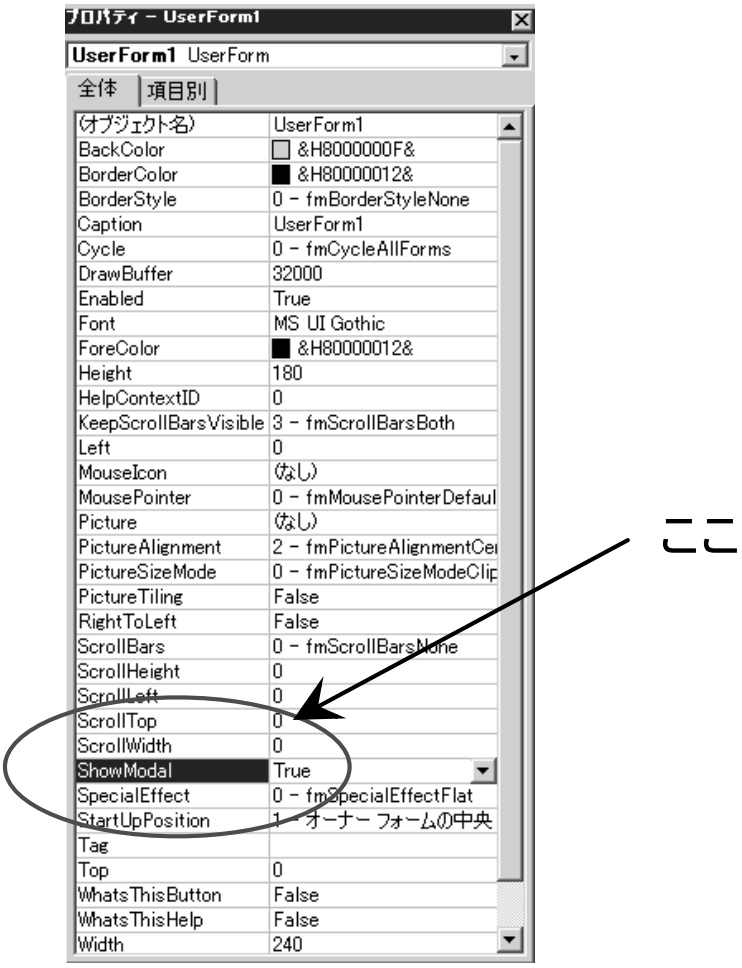

図 6.3: ShowModal を False へ

● 選択範囲から A1 形式のレンジを得る

これは以下のように行います。

```
AdrString = Selection.Address
```

● TextBox 内にマウスが入ったときに処理を行う

これは VB でも、同じなのですが、Excel ではこの処理をよく使います。具体的には、範囲をマウスで選択させて、TextBox にマウスが入ったら、その範囲を TextBox のテキストに入れるなどという処理です。

まず UserForm 内の TextBox でクリックすると、自動的にコードウィンドウが現れて、以下のようにコードが書かれています。

```
Private Sub TextBox1_Change()

End Sub
```

普通はここで、関数の中身を書けば良いのですが、Change というイベントではなくて、マウスがそのテキストに入ったらというイベントで処理を実行したいので、イベントを変更します。図の「change」という部分でメニューをプルダウンさせて、望みのイベントを選びます。

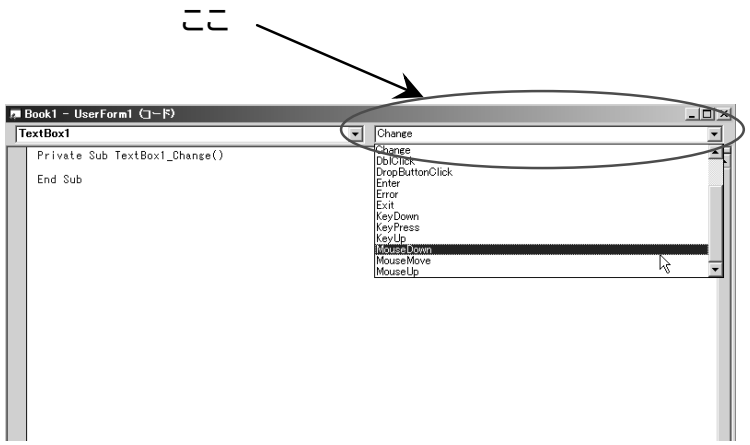

図 6.4: イベントの変更

今、「MouseDown」を選んだとします。するとコードウィンドウに自動的に、対応するプログラムの開始と終了部分が挿入されます。

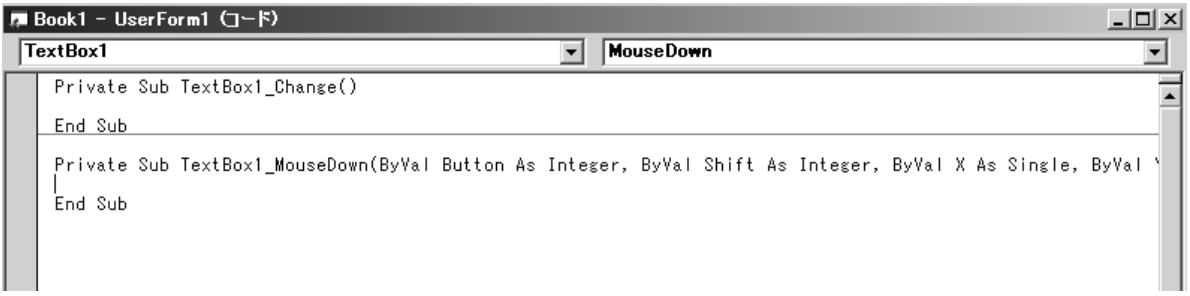

図 6.5: プログラムの枠組みの自動挿入

実際の処理はその中に書いて、最初の

```
Private Sub TextBox1_Change()

End Sub
```

は消します。