5. VBA の応用

§5.1　VBA のデバック

プログラミングでエラーが発生するときがある。VBA には効率的にエラーを検出する機能が備わっている。

Check Point	1. デバッグ機能をマスターする。 2. 各種エラーに対応できる。

5.1.1　エラー処理

プログラムのミスのことを『 バグ (bug : 虫) 』と呼び、そのバグを取り除くことを『 デバッグ (debug) 』という。

① On Error ステートメント

『 On Error 』ステートメントを使うとエラーが発生した場所の処理を指定する。

『 On Error GoTo line 』

< line > には、『 行ラベル 』または『 行番号 』を指定します。実行中にエラーが発生すると指定した < line > 箇所にジャンプしてエラー処理をします。

```
Sub error_process()
  On Error GoTo Err1     ' エラー処理を有効にする
    For i = -10 To 10
      Cells(i, 1) = i
    Next i
  Exit Sub
'---------------------------------------
Err1:
  MsgBox Err.Description  '  エラーメッセージの表示
End Sub
```

図 5.1.1

例えば、『 Cells(i, j) 』 の i, j は自然数でなければならないので、図 5.1.1 のプログラムはエラーが発生する。エラー内容を表示するために『エラーメッセージ』を『メッセージボックス』に表示する。

『 MsgBox 』はメッセージボックスで、『 Err.Description 』はエラー内容を保持するためのプロパティである。『 Err1: 』は、『サブルーチン名 : 』である。

『サブルーチン名』＋『 : 』は、旧式のサブルーチン表記で、引数（ひきすう）が共通になる。

あるステートメント以降にエラー処理を行わないときは、『 GoTo 0 』と指定する。それ以降のエラー処理が無効になり、エラーが発生した場合は処理が無効になる。

② デバック

エラーは以下の３つに分けられる。

『 コンパイル時エラー 』　Visual Basic の文法違反の場合に起こる。

『 実行時エラー 』　存在しないブックを開くなどの違法な操作の場合に起こる。

『 論理エラー 』　処理が意図したように実行されない場合。

VBA には、エラーのデバッグをサポートする次のような機能がある。

・自動構文チェック
・VBA Project のコンパイル
・ブレークポイント
・ステップイン / アウト
・イミディエイトウィンドウ
・ウォッチウィンドウ

5.1.2　自動構文チェック

『 自動構文チェック 』は初期状態で有効になっている。確認をするには、『 ツール(T) 』のプルダウンメニューの『 オプション(O)... 』を選択する。

図 5.1.2 の『 オプション 』ダイアログボックスが開くので、『 編集 』タブの『 自動構文チェック(K) 』がオンになっていることを確認する。

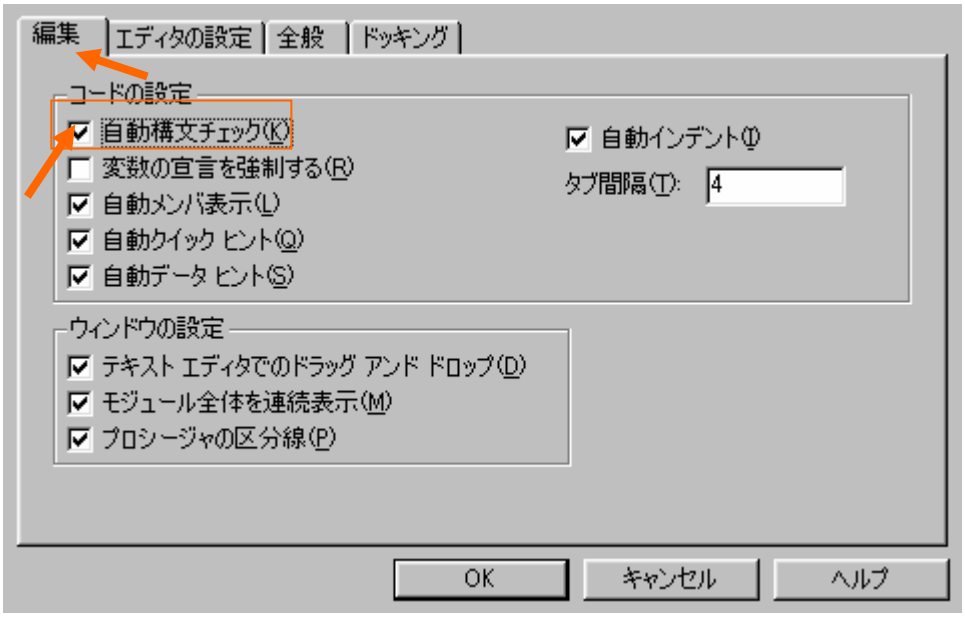


図 5.1.2

変数の宣言を強制する	モジュール単位で変数が必要かどうかの指定。
自動メンバ表示	ポインタのある位置で選択可能な入力項目の一覧表示。 一覧の中から該当する項目を選択して入力できるので、スペルミスの心配がなくなる。
自動クイックヒント	入力した関数とパラメータの情報の表示。
自動データヒント	ポイントした変数の値の表示。デバッグ中は使用できない。

『 自動構文チェック 』をオフにしても構文エラーのある行は赤字表示になる。

5.1.3　デバック機能

実行時にエラーが発生すると、『メッセージボックス』が表示される（図 5.1.3）。

『デバッグ（D）』をクリックするとVBEが開き、プロシージャ中のエラー個所が矢印と黄色で示される（図 5.1.4）。

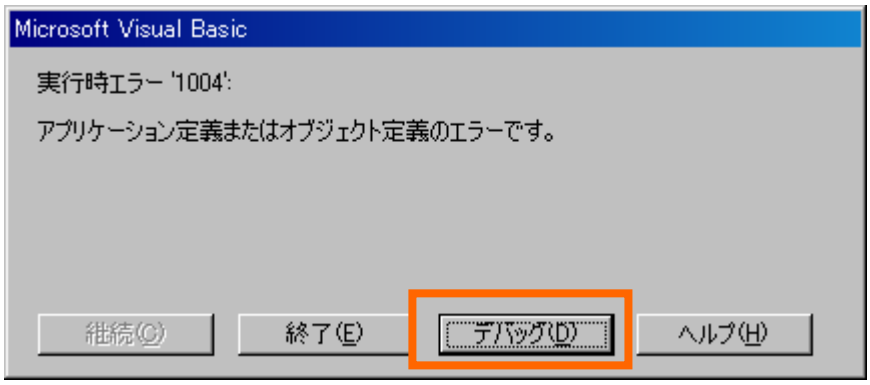


図 5.1.3

このとき、VBA は処理を中断した状態で呼び出される。エラー個所を修正してツールバーの『継続』をクリックすると再実行、『リセット』をクリックすると強制終了できる。

図 5.1.4

『 コンパイル 』とは、VBAコードを実行可能にするための処理である。VBAのメニューから『 デバッグ（D） 』のプルダウンメニューの『 VBAProjectのコンパイル（L） 』を選択すると、コンパイル時にプロジェクト全体の文法違法などをチェックすることが可能となる（図 5.1.5）。

図 5.1.5

5.1.4　ブレークポイント

『 ブレークポイント 』を設定すると、コードをステートメントなどの単位で実行できるため、バグの発見に役立つ。また、プロシージャ実行途中に中断し、変数などの値を目で確認しながらデバッグすることができる。

処理を中断したいプロシージャのステートメントをクリックして、VBEのメニューから『デバッグ（D)』→『ブレークポイントの設定/解除（T)』を選択するとブレークポイントの設定や解除が行える。

画面上の『●』の部分をクリックしても設定や解除が可能である。

この状態でプロシージャを実行すると設定したブレークポイントのステートメントで処理が中断する（矢印と黄色)。

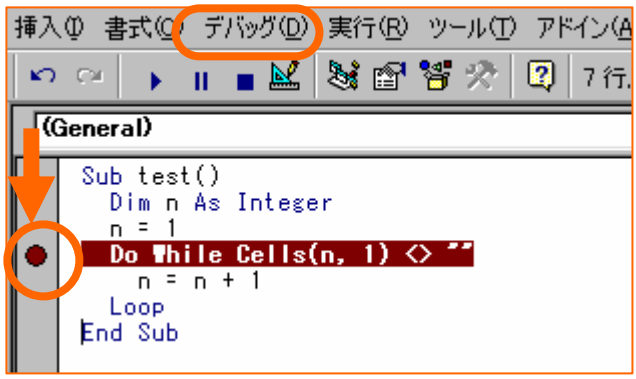


図 5.1.6

注意！

- ブレークポイントは代入ステートメント、及び、実行可能なステートメントで設定する。
  変数の宣言などの宣言ステートメントには設定できない。
- ブレークポイントを設定後にプロシージャを保存しても、ブレークポイントは保存されない。ブックを開きなおすとすべてクリアーされる。

5.1.5　イミディエイト

問題のある箇所や、新規に記述したコードの部分的なテスト、実行中の変数の値を確認、変更することができる。100 回ループ処理を行うプロシージャで、80 回以降のデバッグを行いたい場合、つまり、一気に８０までスキップする場合などに使われる（図 5.1.7）。

```
Sub immediate1()
  For n=1 to 100
    If Cells(n,1).Value="" Then
      Exit For
    End if
    Cells(n,2).Value=Cells(n,1). Value*1.05
    If Cells(n,2).Value>1000 Then
      Cells(n,2).Front.ColorIndex=3
    End If
  Next
End Sub
```

図 5.1.7

図 5.1.7 のプログラム（100 回のループ）で、８０回目のループから実行するには、

① VBEメニューから『 表示(V) 』→ 『 イミディエイトウィンドウ(I) 』を選択する。
② 『 ステップイン 』で 3 行目まで実行する。
③ イミディエイトウィンドウ（図 5.1.8）内に『 n=80 ↲ 』と入力します。必ず『 ↲ 』キーを押して改行する。
④ プロシージャを実行する。

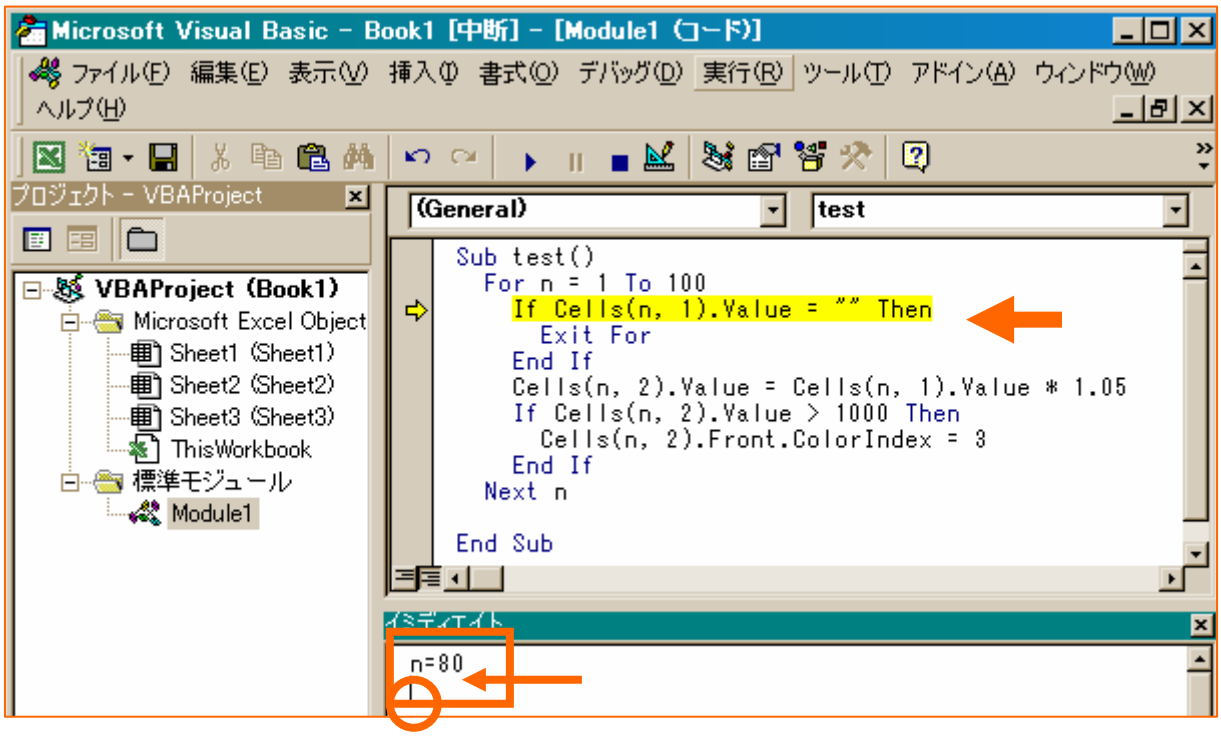


図 5. 1. 8

5.1.6　ウォッチウィンドウ

実行中の変数の内容を監視しながら処理を進める。

a. 変数に代入された値の変化をウォッチする場合

① プロシージャ内に監視したい変数名の範囲を指定する（図 5.1.9）。

② VBEメニューから『 表示(V) 』→ 『 ウォッチ式の追加(A)... 』を選択する。

③ 『 ウォッチ式の追加 』ダイアログボックスが開くので『 式(E): 』蘭に手順①で指定した変数が表示されていることを確認して『 OK 』をクリックする（図 5.1.10）。

手順①～③を繰り返せば、複数の変数をウォッチウィンドウに割り当てることもできる。

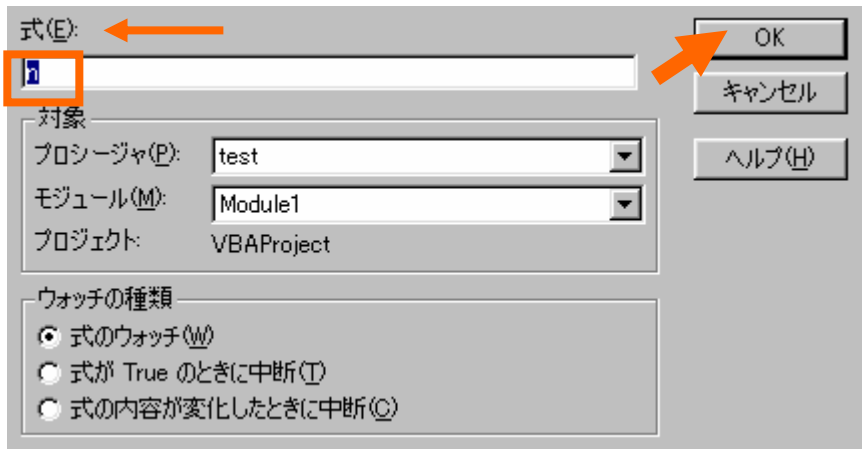


図 5.1.9

図 5.1.10

④ プロシージャ内の先頭行にカーソルを置き、VBEのメニューから『 デバック(D) 』→ 『 ステップイン(I) 』を選択する。

これでステップインを実行するごとにウォッチウィンドウの値が変化する。また、プロシージャ内の任意の行にブレークポイントを設定してステップインを行えば、その地点からの変数をウォッチすることが可能になる。

- 値　式の値を示します。ただし、その式の対象のプロシージャが実行されていないときは
  『 ＜対象範囲外＞ 』
  となり、値は表示されません。
- 型　変数や式の種類を表示します。
- 対象　モジュール名とプロシージャ名などの式の対象範囲を示します。

b.ウォッチ式に条件式を代入する場合

For ステートメントの実行途中で、『 変数 m 』のカウントが 5 未満の時の『 変数 x 』や『 変数 n 』の経過をウォッチウィンドウで知ることができる。

ウォッチウィンドウの式に『 **m < 5** 』という条件式を設定する（図 5.1.12）。

```
Sub ウォッチ式()
  Dim x As Integer
  x = 0
  For n = 1 To 10
    For m = 1 To 10
      Cells(n, m).Value = 0
      x = x + m
    Next m
  Next n
End Sub
```

図 5.1.11

m < 5 のとき、ウォッチウィンドウの値に『 **True** 』が表示され、m ≧ 5 のときは、『 **False** 』が表示される（図 5.1.3）。

図 5.1.12

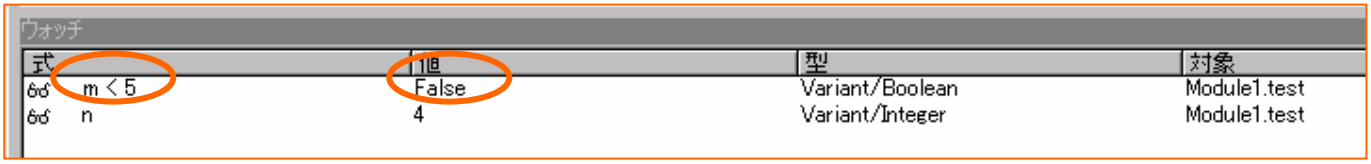


図 5.1.13

§5.2　コントロール

コントロールとはボタンや入出力ウィンドウなどのオブジェクトのことで、例えば、ボタンをクリックすることで処理を行う。オブジェクトを「ユーザーフォーム」上に自由にレイアウトして、ユーザーのオリジナルな画面を作製することができる。

Check Point	1. ユーザーフォームとコントロールの関係を理解する。 2. プロパティによって、コントロールの各々の値を設定する。

5.2.1　ユーザーフォームの設定

①　ユーザーフォームの挿入

『挿入（I）』のプルダウンメニューの『ユーザーフォーム（U）』を選択する（図 5.2.1）。

図 5.2.2 の『UserForm1』とコントロールの『ツールボックス』が現れる。

図 5.2.1

②　コマンドボタン

『ツールボックス』のコマンドボタン　をクリックして　『UserForm1』上の適当な位置でドラッグする（図 5.2.2）。図 5.2.3 のような『CommandButton1』が作成される。ドラッグした大きさでコマンドボタンの大きさが決まる。作成された『CommandButton1』をクリックして、配置を変えたり、カーソルをコマンドボタンの端に持っていき、自由に大きさを変えたりすることができる。

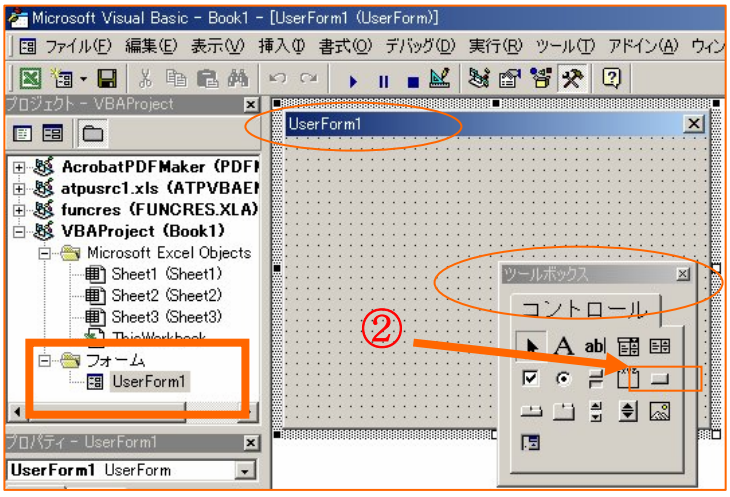


図 5.2.2

③　テキストボックス

『ツールボックス』のテキストボックスボタン　ab|　をクリックして『UserForm1』上の適当な位置でドラッグする（図 5.2.3）。テキストボックスが現れる（図 5.2.4）。

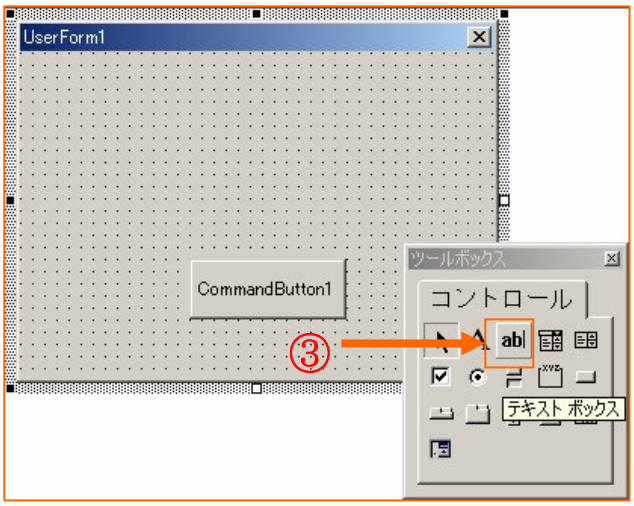


図 5.2.3

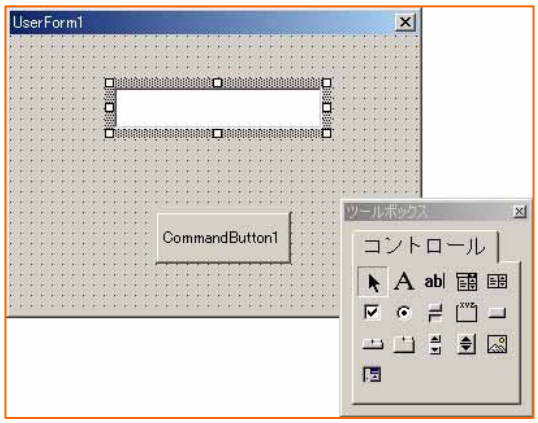


図 5.2.4

④　オブジェクトのコード表示

『UserForm1』上にある『CommandButton1』をダブルクリックする。図 5.2.5 のプロシージャが表示される。

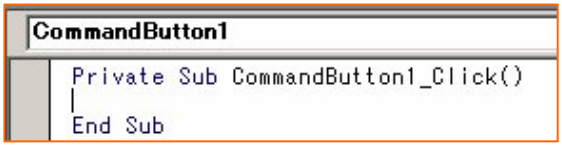


図 5.2.5

さらに、このプロシージャに図 5.2.6 のようなプログラムを入力する。

『CommandButton1』をクリックするとテキストボックスにランダムな整数（0～99）を表示するプログラムである。

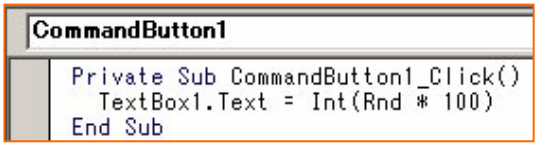


図 5.2.6

プログラムを実行すると図 5.2.7 のようなウィンドウが立ち上がり、『CommandButton1』をダブルクリックするたびに、ランダムな整数が出力される。

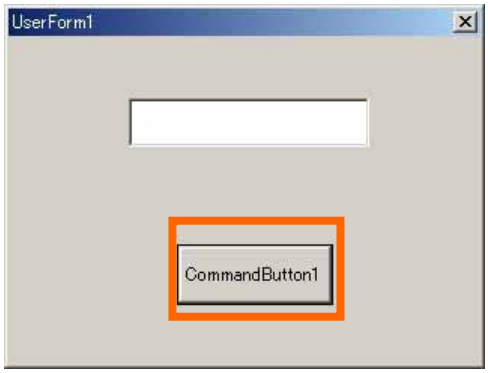


図 5.2.7

5.2.2　プロパティの変更

①　プロパティ

『CommandButton1』ボタンの名前、フォント、色、大きさなどを変えたい場合がある。それぞれのオブジェクトにはプロパティがあり、これを変更することによって簡単にカスタマイズすることができる。

②　ボタンの表示文字の変更

『UserForm1』上にある『CommandButton1』をクリックする。『プロパティ』が『CommandButton1』になる（図 5.2.8）。

『Caption』を選択して右側のテキストボックスに『実行』と入力する。

『CommandButton1』から『実行』に変更される。

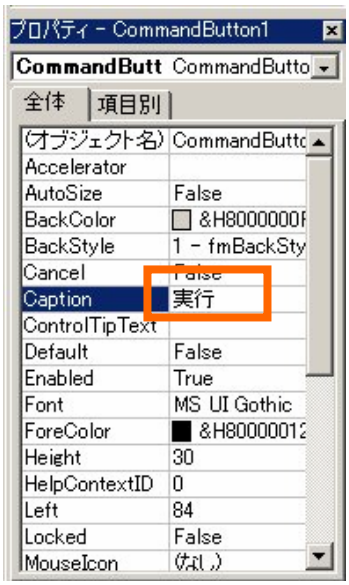
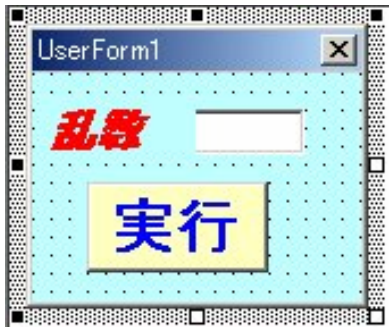


図 5.2.8

課題 1.　プロパティを変更して下の図のような実行画面を作成しなさい！



5.2.3　簡単なじゃんけんゲーム

①　自分が出す手

まず、コマンドボタンを 3 つ作る。それぞれのボタンをクリックすると、テキストボックスに『グー』『チョキ』『パー』と表示させるプログラムは図 5.2.9 のようになる。

この 3 つのボタンを押すことによって自分の出す手を決める。

```
Private Sub CommandButton1_Click()
  TextBox1.Text = "グー"
End Sub
Private Sub CommandButton2_Click()
  TextBox1.Text = "チョキ"
End Sub
Private Sub CommandButton3_Click()
  TextBox1.Text = "パー"
End Sub
```

図 5.2.9

②　コンピュータが出す手

次にコンピュータがランダムに出す手を決めるプログラムを作る。4 つめのコマンドボタン『**CommandButton4**』を作成する（図 5.2.10）。プログラムの 1 例を図 5.2.11 に示す。

自分の手を決めていないとコンピュータはじゃんけんをしないで『何を出すか決めてください！』と表示する。これは、『変数 jan』の値によって判断される。

> グローバル変数
>
> 『変数 jan』はプログラムの最初に指定されているのでグローバル変数である。ローカル変数とグローバル変数の違いは 4.5 節のプロシージャで説明している。

乱数で発生させられる 0～99 の整数を 3 で割ってそのあまりによって『グー』『チョキ』『パー』を判断させる。それぞれ 1/3 の確率になる。

コンピュータの手を『**TextBox2**』に表示させる。

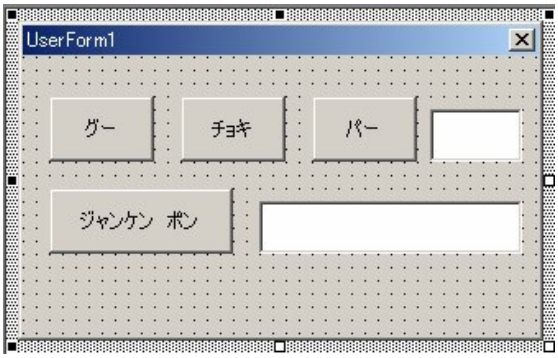


図 5.2.10

```
Dim jan As Integer
Private Sub CommandButton1_Click()
  TextBox1.Text = "グー"
  jan = 1
End Sub
Private Sub CommandButton2_Click()
  TextBox1.Text = "チョキ"
  jan = 2
End Sub
Private Sub CommandButton3_Click()
  TextBox1.Text = "パー"
  jan = 3
End Sub
Private Sub CommandButton4_Click()
  If jan = 0 Then
    TextBox2.Text = "何を出すか決めてください！"
    Beep
  Else
    n = Int(100 * Rnd)
    ken = n Mod 3
    Select Case ken
      Case 0
        TextBox2.Text = "グー"
      Case 1
        TextBox2.Text = "チョキ"
      Case 2
        TextBox2.Text = "パー"
    End Select
  End If
End Sub
```

図 5.2.11

③　勝ち負けの判断

実際のプログラムの一例を図 5.2.12 に示す。勝敗の判断を『**Select Case**』を使ってしているが、他のアルゴリズムも考えられる。

```
Private Sub CommandButton4_Click()
  If jan = 0 Then
    TextBox2.Text = "何を出すか決めてください！"
    Beep
  Else
    n = Int(100 * Rnd)
    ken = n Mod 3
    Select Case ken
      Case 0
        TextBox2.Text = "グー"
        Select Case jan
          Case 1
            TextBox3.Text = "あいこ"
          Case 2
            TextBox3.Text = "あなたの負けです"
          Case 3
            TextBox3.Text = "あなたの勝ちです"
        End Select
      Case 1
        TextBox2.Text = "チョキ"
        Select Case jan
          Case 1
            TextBox3.Text = "あなたの勝ちです"
          Case 2
            TextBox3.Text = "あいこ"
          Case 3
            TextBox3.Text = "あなたの負けです"
        End Select
       Case 2
        TextBox2.Text = "パー"
        Select Case jan
          Case 1
            TextBox3.Text = "あなたの負けです"
          Case 2
            TextBox3.Text = "あなたの勝ちです"
          Case 3
            TextBox3.Text = "あいこ"
        End Select
    End Select
  End If
End Sub
```

課題 1.　他の勝敗判断のプログラムを作れ！

課題 2.　簡単なゲームのプログラムを作れ！