—ダウンロード特典—
VBA開発支援ツール
階層化フォーム

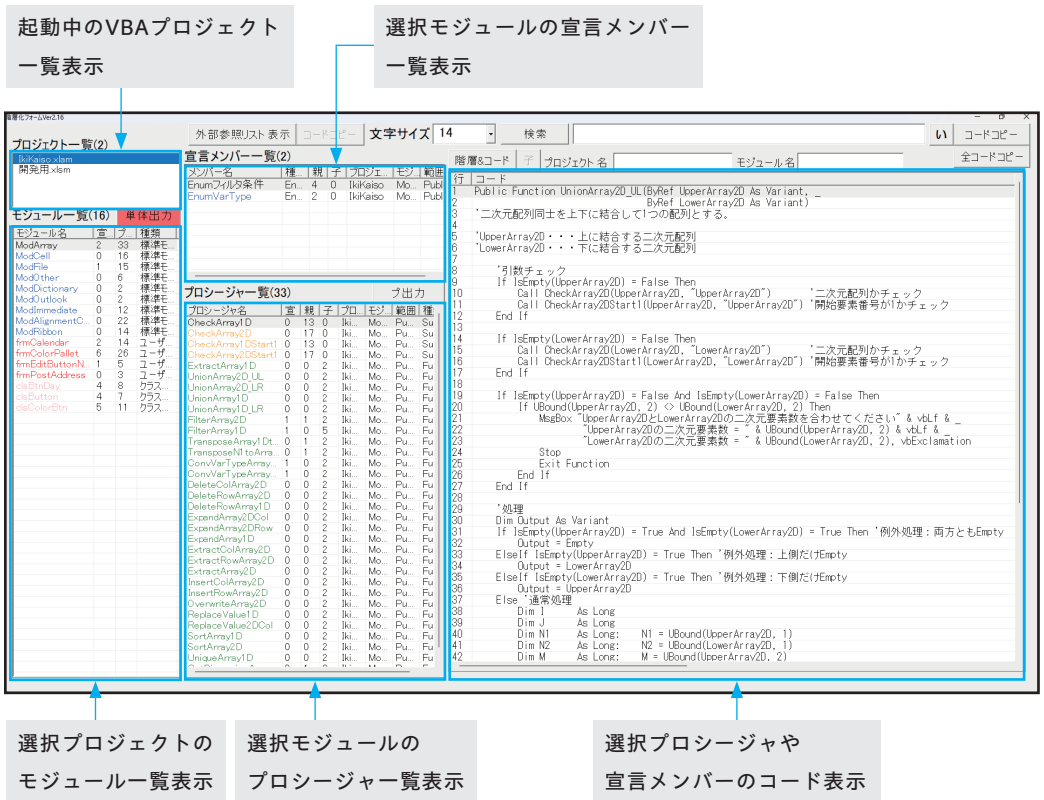
付録2

階層化フォームの使い方

ここでは「階層化フォーム」の
実際の使用法を解説します。
「VBAプロジェクト」「モジュール」「宣言メンバー」「プロシージャ」を
一覧表示して検索したり、依存関係を階層構造で表示するほか、
「外部参照プロシージャー覧取得」など実に多機能なツールです。
みなさんのVBAの開発効率が信じられないレベルで向上しますので、
ぜひとも積極的に階層化フォームを活用してください。

付 2 - 1 全体構造

「付録1　階層化フォームの基本設定」でリボンに登録した［階層化フォーム起動］ボタンをクリックすると、次のような「階層化フォーム」が表示されます。



では、それぞれのウィンドウについて、まずは簡単に解説します。

●プロジェクト一覧

起動中のプロジェクト（VBAProject）のWorkbook名が一覧で表示されます。

●モジュール一覧

プロジェクト一覧にて選択したプロジェクトの中のモジュールが一覧で表示されます。

このとき、モジュールの種類（ThisWorkbookオブジェクト、Worksheetオブジェクト、ユーザーフォーム、標準モジュール、クラスモジュール）は色分けして表示されます。

また、モジュール名のほかに項目として「宣言メンバー個数」「プロシージャ個数」「種類」が表示されます。

●宣言メンバー一覧

モジュール一覧で選択したモジュールの中に含まれる宣言メンバー一覧が表示されます。

「宣言メンバー」とは、モジュールの冒頭に宣言するWindowsAPI、ユーザー定義型（Type）、Enum、定数（Const）、グローバル変数（Private、Public、Dim）の総称です。そして、これらの種類は色分けして表示されます。

また、メンバー名のほかに項目として「種類」「親となる宣言メンバー個数」「子となる宣言メンバー、プロシージャ個数」「所属プロジェクト名」「所属モジュール名」「範囲」が表示されます。

●プロシージャ一覧

モジュール一覧で選択したモジュールの中に含まれるプロシージャ一覧が表示されます。

このとき、プロシージャはSubプロシージャとFunctionプロシージャの２種類で色分けして表示されます。

また、プロシージャ名のほかに項目として「使用している宣言メンバー個数」「親となるプロシージャ個数」「子となるプロシージャ個数」「所属プロジェクト」「所属モジュール」「範囲」「種類」が表示されます。

●コード表示

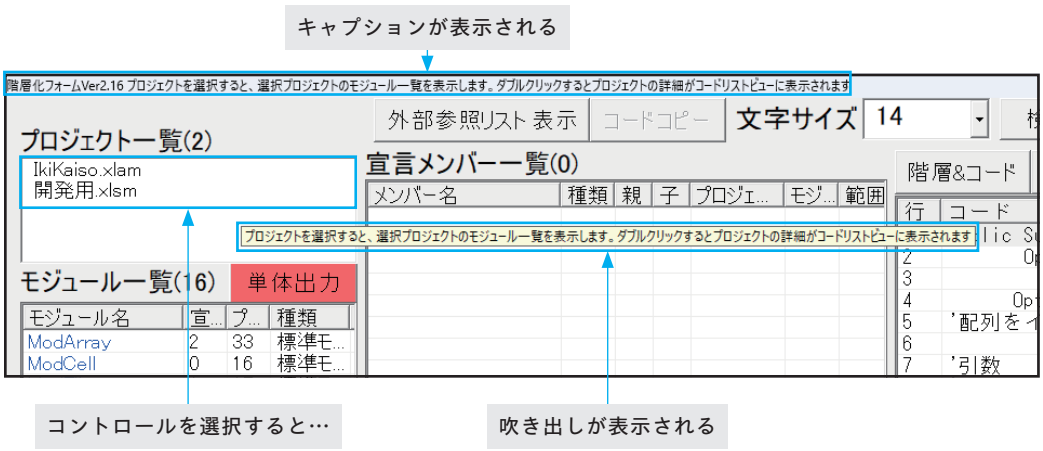
宣言メンバー一覧で選択した宣言メンバー、もしくはプロシージャ一覧で選択したプロシージャのコードが表示されます。

「モジュール一覧」「宣言メンバー一覧」「プロシージャ一覧」は、リストビューのコントロールを用いていますので、ヘッダーの項目をダブルクリックするとその項目で並び替えができます。

次節以降、階層化フォームの機能をより具体的に紹介していきます。

付 2 - 2 キャプションや吹き出しで使い方を表示する

まず、各コントロールにマウスカーソルをあてると、そのコントロールの使い方の説明がキャプションや吹き出しで表示されます。

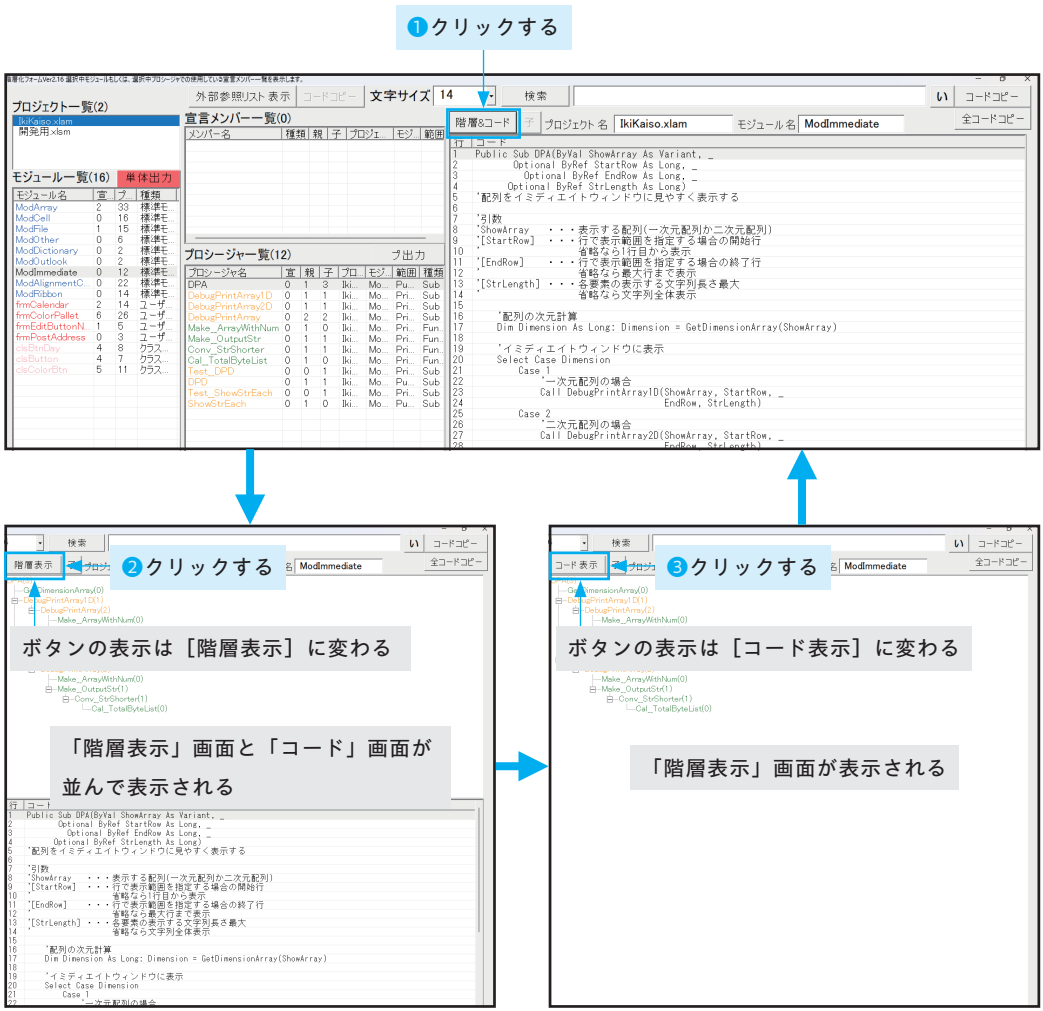


付 2 - 3 階層構造を表示する

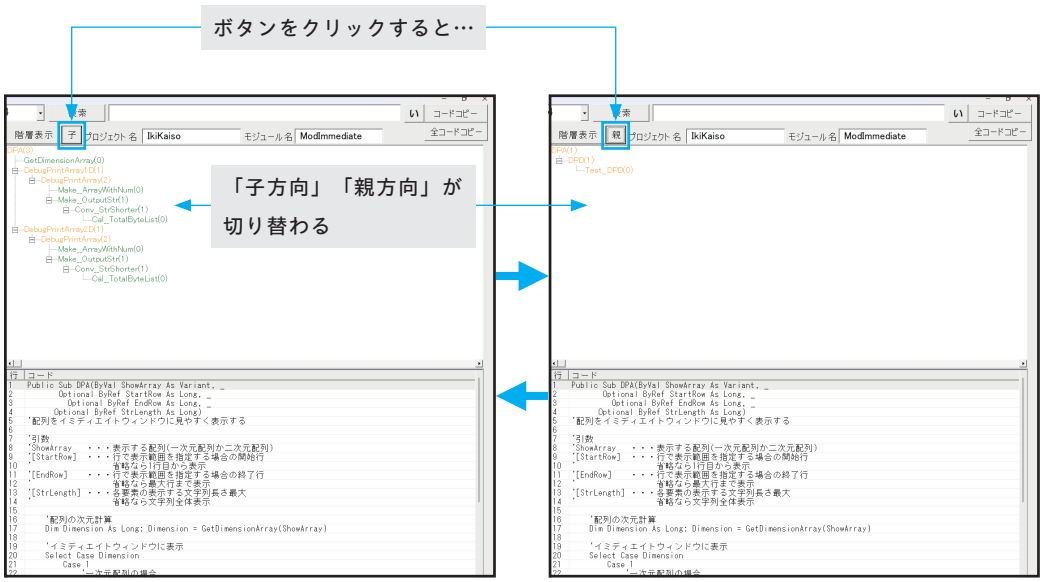
［階層＆コード］ボタンをクリックすると、「コード」と「階層表示」が縦に並べて表示されます。

このとき、ボタンの表示は「**階層表示**」に変わっているので、もう1回クリックすると「階層構造」だけが表示されます。

次に、ボタンの表示は「**コード表示**」に代わっているのでもう1回クリックすると「コード」だけが表示されます。



また、ボタンの右にある［子］ボタンをクリックすると、「階層構造」の表示を「子方向」「親方向」に切り替えることができます。

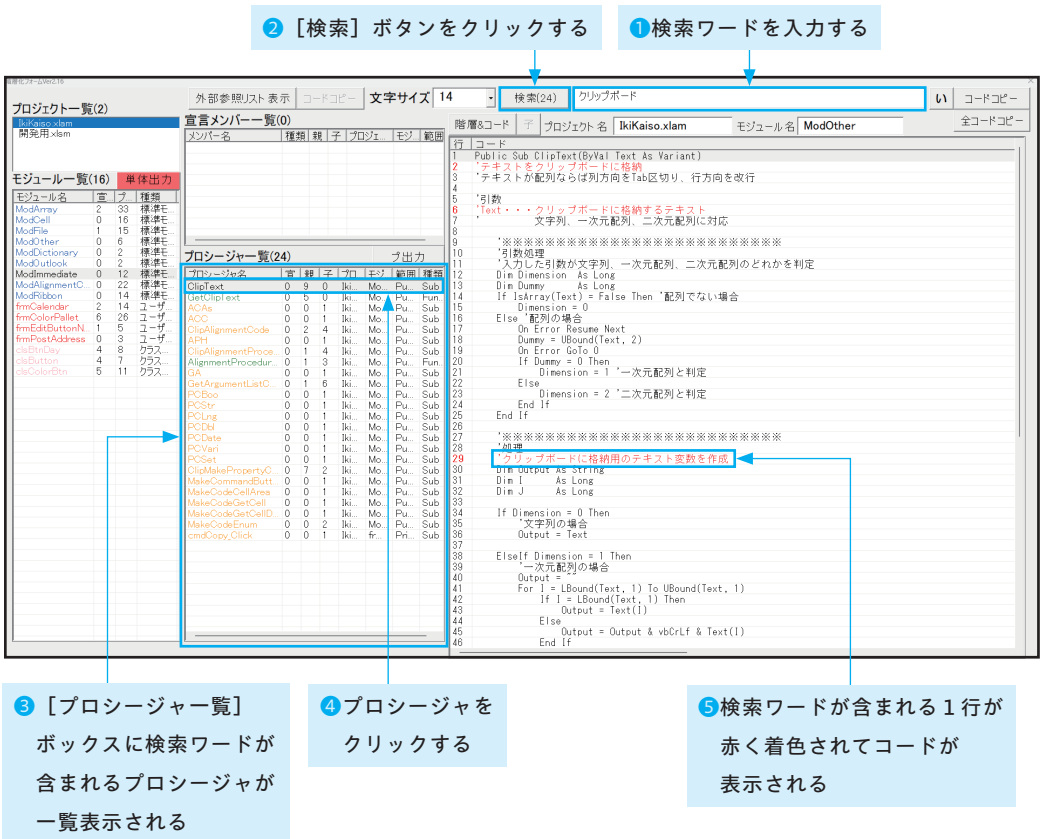


付 2 - 4 検索機能

画面右上のテキストボックスに検索ワードを入力して**［検索］ボタン**をクリックすると、**［プロシージャー覧］ボックス**に検索ワードが含まれるプロシージャの一覧が表示されます。

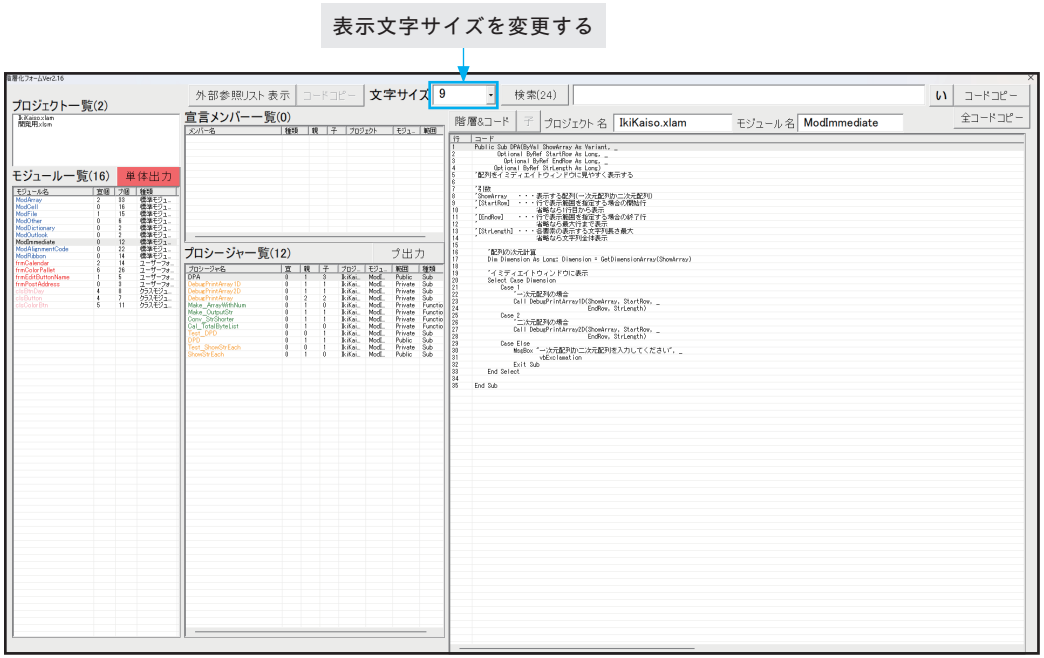
また、［プロシージャー覧］ボックスでプロシージャを選択するとコードが表示され、検索ワードが含まれる行が赤く着色されます。

この機能は、開発用アドインに構築した汎用プロシージャを思い出すときに威力を発揮します。



付 2 - 5 表示文字サイズを変更する

「文字サイズ」の隣の数字を変更することで全体の表示文字サイズを変更することができます。PCの画面サイズに合わせて見やすいように切り替えてください。

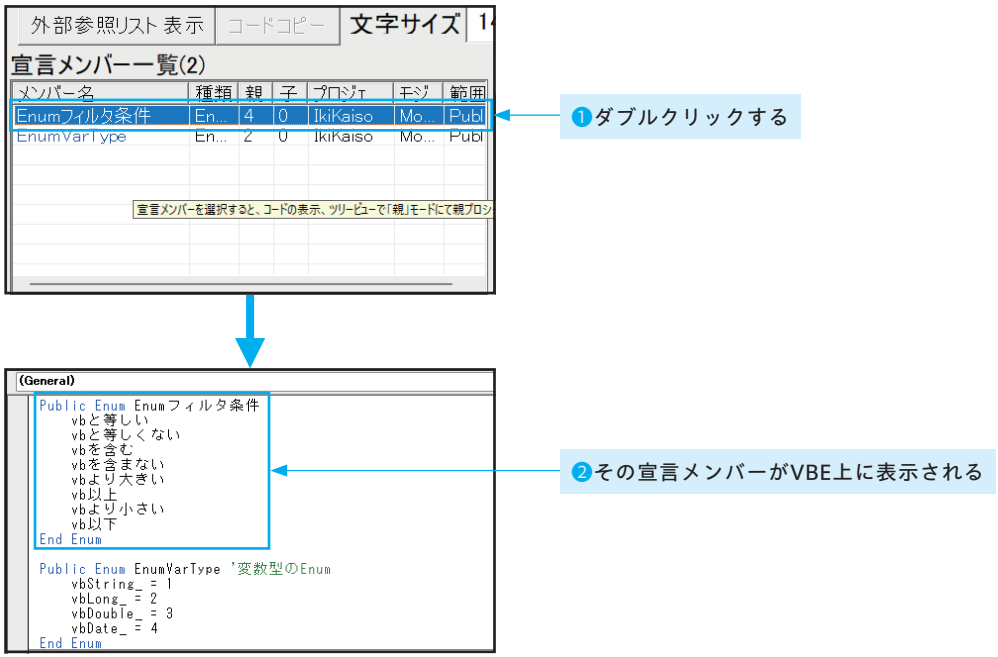


付 2 - 6 ダブルクリックでコードを表示する

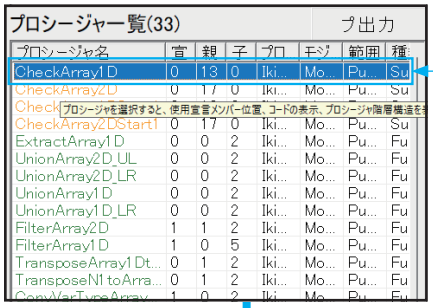
宣言メンバー一覧、プロシージャ一覧、階層構造における宣言メンバー、プロシージャをダブルクリックするとVBEのコードウィンドウ上での位置に自動的にジャンプします。

そのため、実際のコードをVBEですぐに確認できるようになっています。

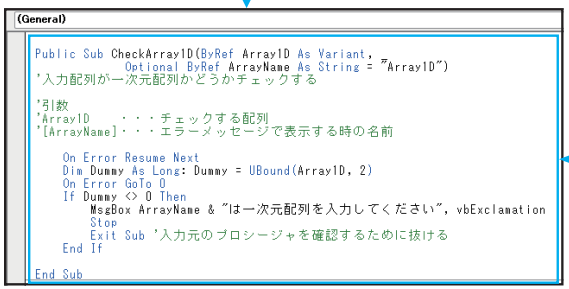
●宣言メンバー一覧から宣言メンバーを表示する



●プロシージャー覧からプロシージャを表示する

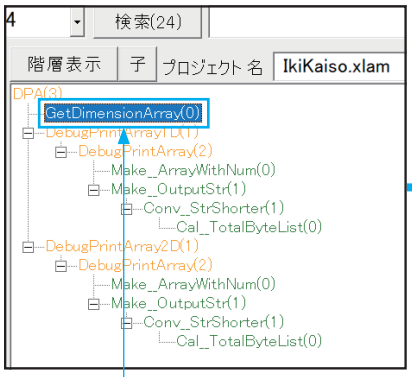


❶ダブルクリックする

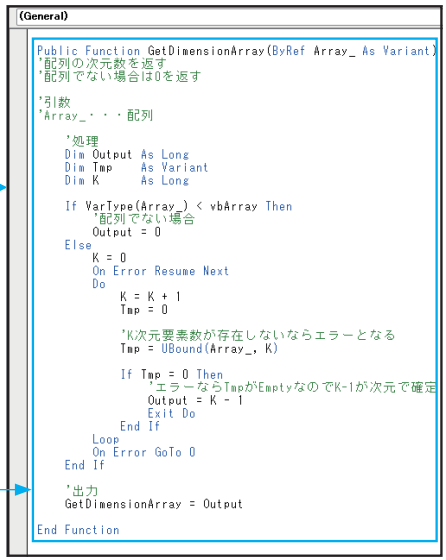


❷そのプロシージャが
VBE上に表示される

●階層表示からプロシージャを表示する



❶ダブルクリックする

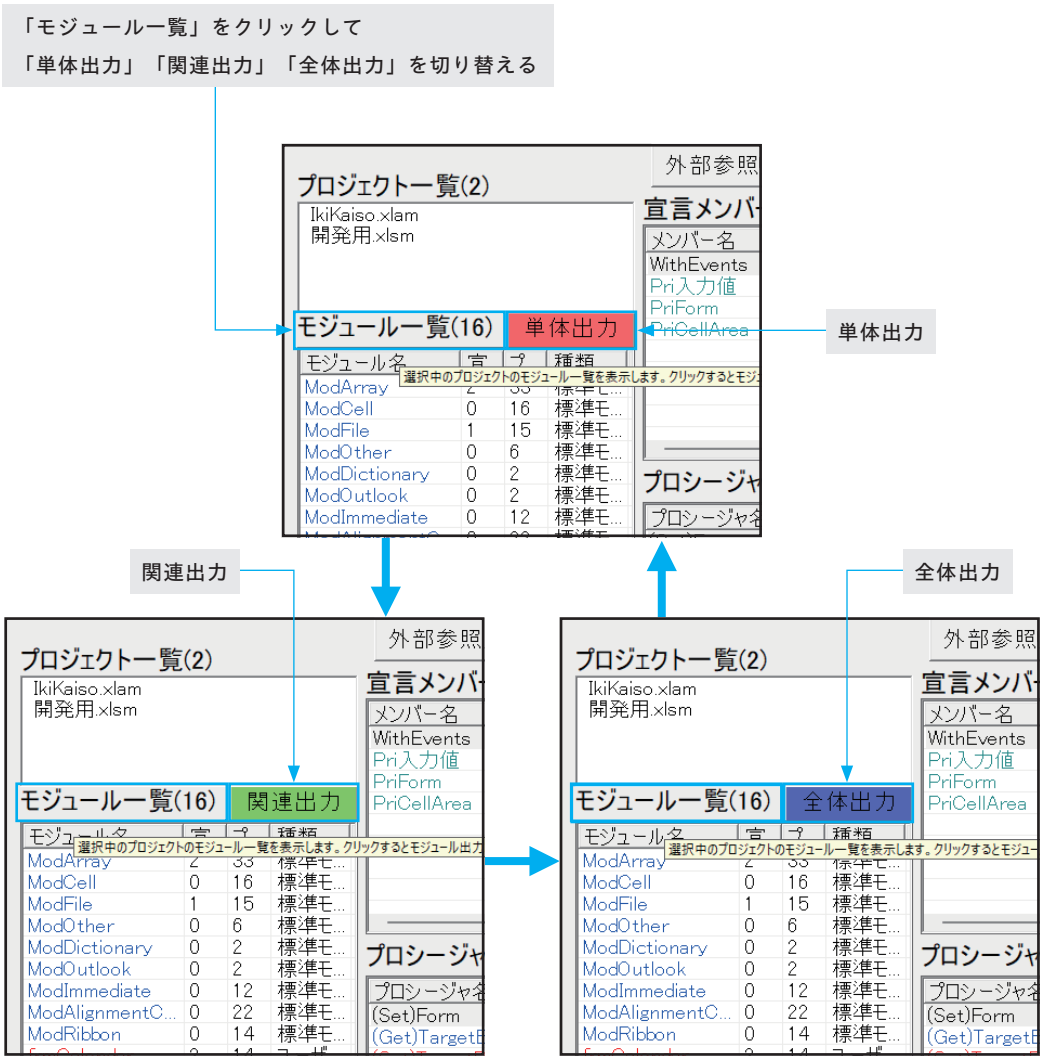


❷そのプロシージャが
VBE上に表示される

付 2 - 7 モジュールを出力する

モジュール一覧に表示されているモジュールを3種類の方法で出力（エクスポート）します。

3種類の出力方法は「**単体出力**」「**関連出力**」「**全体出力**」で、出力方法は「**モジュール一覧**」のラベルをクリックすることで切り替えることができます。



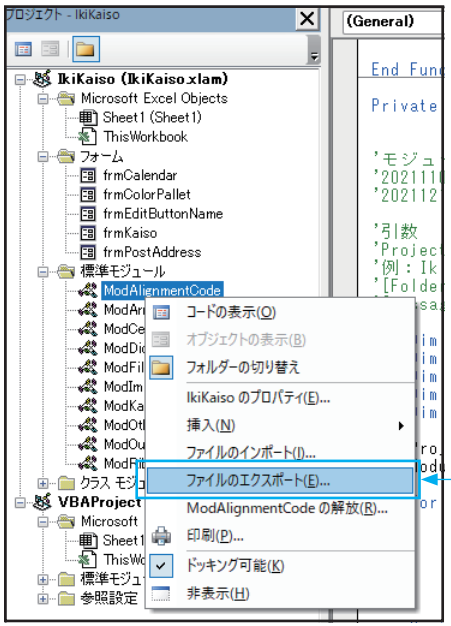
では、この3種類の出力方法について解説します。

●単体出力

モジュールを通常どおりに出力します。

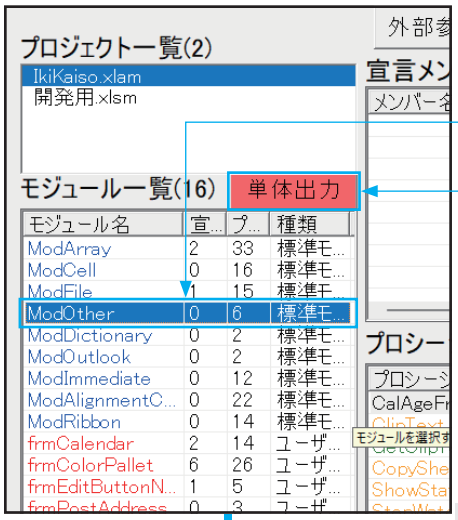
ユーザーフォームは「frmファイル」および「frxファイル」、標準モジュールは「basファイル」、クラスモジュールは「clsファイル」が出力されます。

この「単体出力」は、VBEでモジュールを右クリックして表示されるメニューから［ファイルのエクスポート（E）...］を実行するのと同じ機能です。

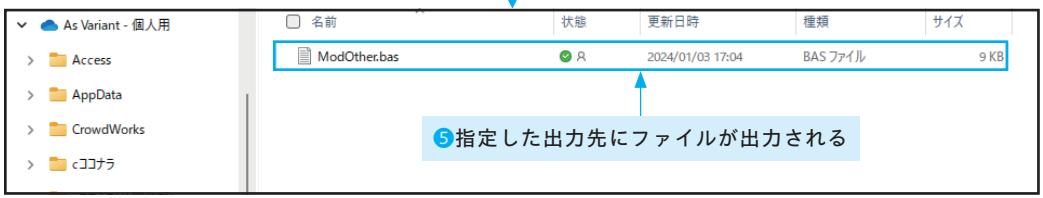


右クリックして［ファイルのエクスポート（E）...］を実行するのと同じ機能

次の手順で操作してください。



［出力先フォルダを選択してください］画面が表示される



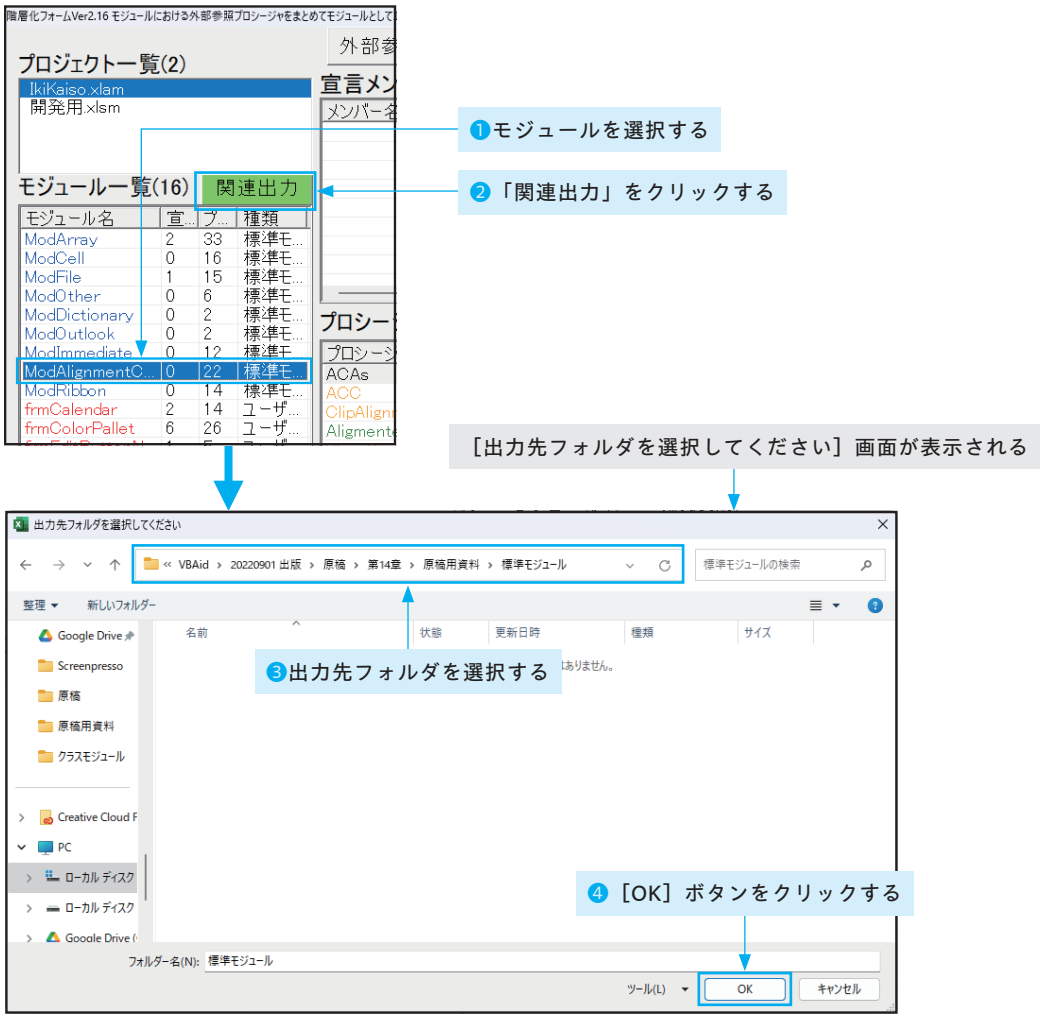
●関連出力

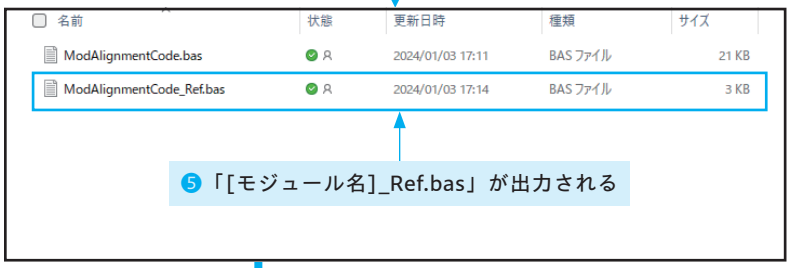
モジュール内のプロシージャが他のモジュールから参照しているプロシージャ（外部モジュール参照プロシージャ）を一覧でまとめたものを出力します。

出力形式は、basファイルで、VBEでプロジェクトに読込（インポート）すると標準モジュールとして読み込まれます。

「単体出力」の場合、外部モジュール参照プロシージャが含まれていないため、出力されたファイル単体では機能しませんが、この「関連出力」で出力された「basファイル」と一緒にプロジェクトにインポートすることで機能するようになります。

次の手順で操作してください。





ModAlignmentCode_Ref.bas

ファイル　編集　表示

```
Attribute VB_Name = "ModAlignmentCode_Ref"
Option Explicit

'GetClipText･･･元場所：IkiKaiso.ModOther
'ClipText   ･･･元場所：IkiKaiso.ModOther



Private Function GetClipText() As String
'クリップボードに格納中の文字列データを取得する
'「Microsoft Forms 2.0 Object Library」ライブラリを参照すること

    '処理
    'クリップボードに格納されているのが画像以外の場合のエラー回避
    On Error Resume Next
    Dim Output As String
    Dim Clip   As New DataObject
    With Clip
        .GetFromClipboard
        Output = .GetText
    End With
    On Error GoTo 0

    '出力
    GetClipText = Output

End Function

Private Sub ClipText(ByVal Text As Variant)
'テキストをクリップボードに格納
'テキストが配列ならば列方向をTab区切り、行方向を改行

'引数
'Text･･･クリップボードに格納するテキスト
'           文字列、一次元配列、二次元配列に対応

    '※※※※※※※※※※※※※※※※※※※※※※※※※※※※
    '引数処理
    '入力した引数が文字列、一次元配列、二次元配列のどれかを判定
    Dim Dimension  As Long
    Dim Dummy      As Long
    If IsArray(Text) = False Then '配列でない場合
        Dimension = 0
    Else '配列の場合
        On Error Resume Next
```

内容は他モジュールで参照している
プロシージャー覧

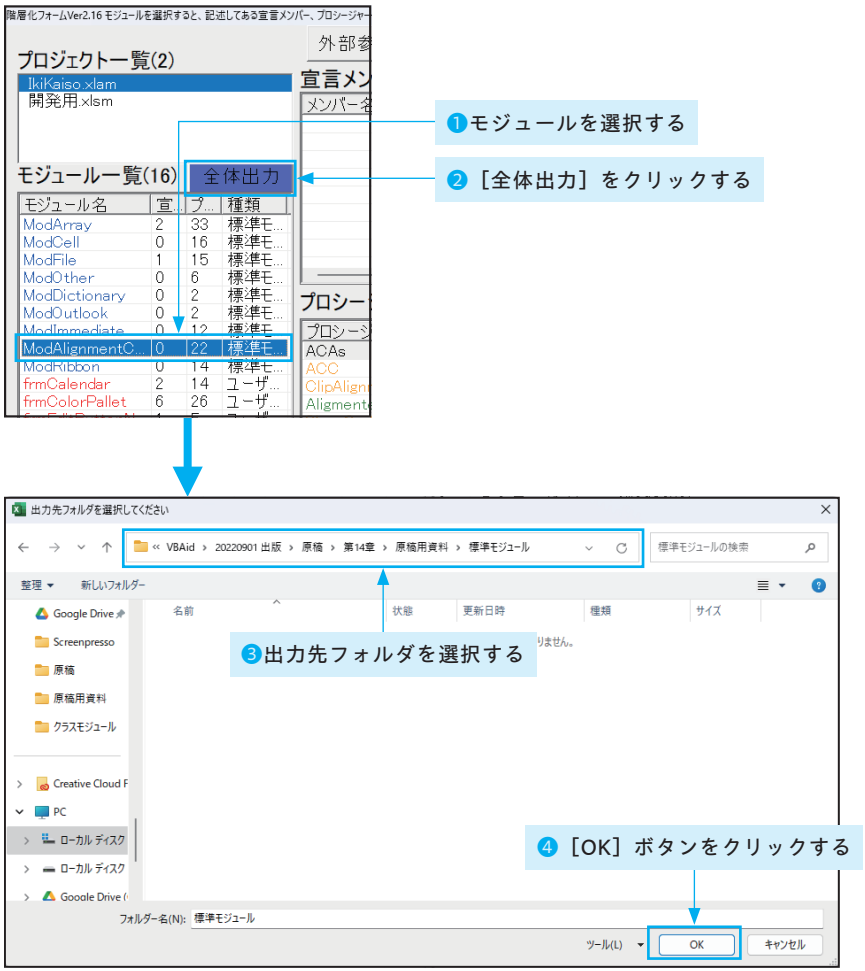
付録2
階層化フォームの使い方

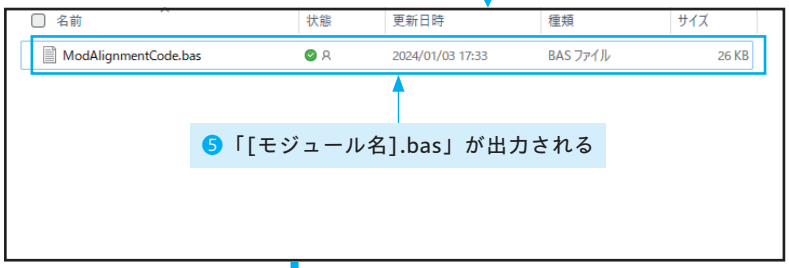
●全体出力

モジュール内のコードおよび、外部モジュール参照プロシージャの2つをまとめて1つの「basファイル」として出力します。

「単体出力」で出力したファイルは、「関連出力」で出力した「basファイル」と一緒にプロジェクトにインポートしないと動作しませんでしたが、「全体出力」はまとまった「basファイル」として出力されるため、このファイルだけをインポートするだけでインポート先のプロジェクト内で動作するようになります。

ただし、1つの「basファイル」内にプロシージャをすべて含めることは可能ですが、参照しているクラスモジュールは含めることはできないので、出力するモジュールが参照しているクラスモジュールは別途出力させる必要があります。





ModAlignmentCode.bas

ファイル　編集　表示

```
Attribute VB_Name = "ModAlignmentCode"
Option Explicit

'イミディエイトウィンドウを活用してコードの整形

'ACAs                        ･･･元場所：IkiKaiso.ModAlignmentCode
'ACC                         ･･･元場所：IkiKaiso.ModAlignmentCode
'ClipAlignmentCode           ･･･元場所：IkiKaiso.ModAlignmentCode
'AligmentedArray2D           ･･･元場所：IkiKaiso.ModAlignmentCode
'ShowCodeWindow              ･･･元場所：IkiKaiso.ModAlignmentCode
'APH                         ･･･元場所：IkiKaiso.ModAlignmentCode
'ClipAlignmentProcedureHead･･･元場所：IkiKaiso.ModAlignmentCode
'AlignmentProcedureHead      ･･･元場所：IkiKaiso.ModAlignmentCode
'Conv__ConnectUnderbar       ･･･元場所：IkiKaiso.ModAlignmentCode
'GetStrInParentheses         ･･･元場所：IkiKaiso.ModAlignmentCode
'Modify__ArgumentHead        ･･･元場所：IkiKaiso.ModAlignmentCode
'GA                          ･･･元場所：IkiKaiso.ModAlignmentCode
'GetArgumentListComment      ･･･元場所：IkiKaiso.ModAlignmentCode
'Get__ArgumentName           ･･･元場所：IkiKaiso.ModAlignmentCode
'PCBoo                       ･･･元場所：IkiKaiso.ModAlignmentCode
'PCStr                       ･･･元場所：IkiKaiso.ModAlignmentCode
'PCLng                       ･･･元場所：IkiKaiso.ModAlignmentCode
'PCDbl                       ･･･元場所：IkiKaiso.ModAlignmentCode
'PCDate                      ･･･元場所：IkiKaiso.ModAlignmentCode
'PCVari                      ･･･元場所：IkiKaiso.ModAlignmentCode
'PCSet                       ･･･元場所：IkiKaiso.ModAlignmentCode
'ClipMakePropertyCode        ･･･元場所：IkiKaiso.ModAlignmentCode
'GetClipText                 ･･･元場所：IkiKaiso.ModOther
'ClipText                    ･･･元場所：IkiKaiso.ModOther



Public Sub ACAs()
'AlignmentCodeAsの略
'クリップボードに格納させた変数宣言コードを
'「As」で整列して、クリップボードに格納する

    Call ClipAlignmentCode(" As ")

End Sub

Public Sub ACC()
'AlignmentCodeCommentの略
'クリップボードに格納させたコードの
```

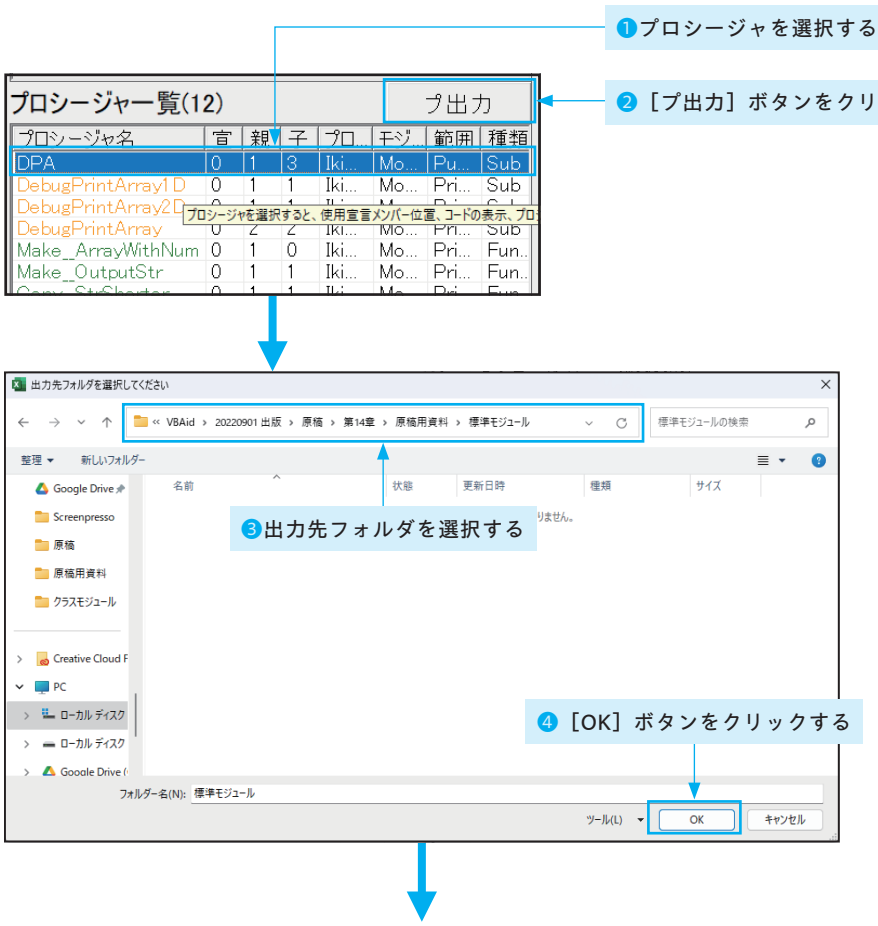
内容は使用しているプロシージャをすべて含めたもの

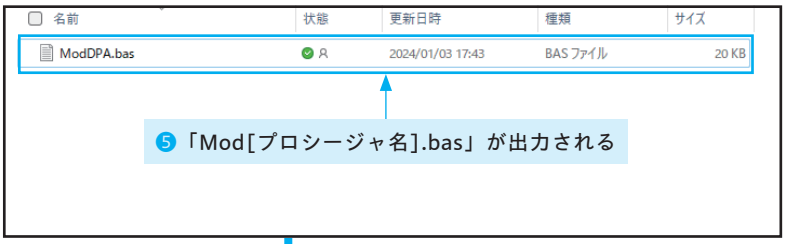
付 2 - 8 プロシージャの「basファイル」を出力する

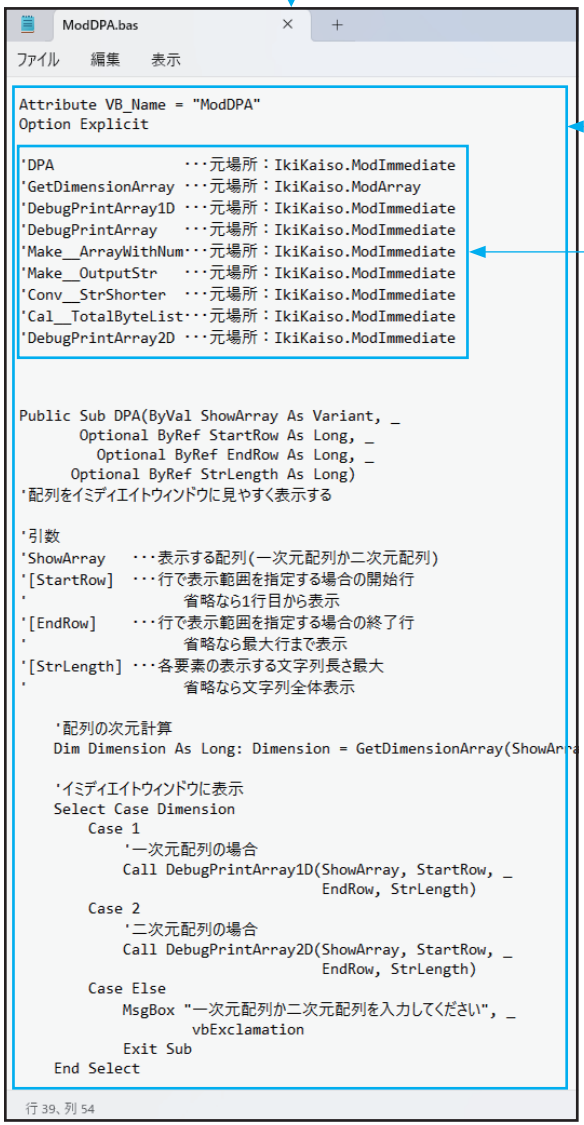
プロシージャ一覧の右にある［プ出力］ボタンは、選択プロシージャを別で参照しているプロシージャ（子プロシージャ）も含めてまとめて「basファイル」として出力する機能です。

選択プロシージャのコード単体では機能せず、子プロシージャも含めたコードでようやく機能します。

［プ出力］ボタンで出力した「basファイル」は、プロジェクトにインポートすると標準モジュールとして読み込まれ、選択プロシージャが使えるようになります。

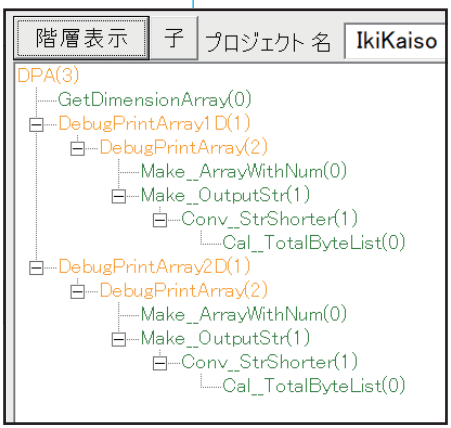






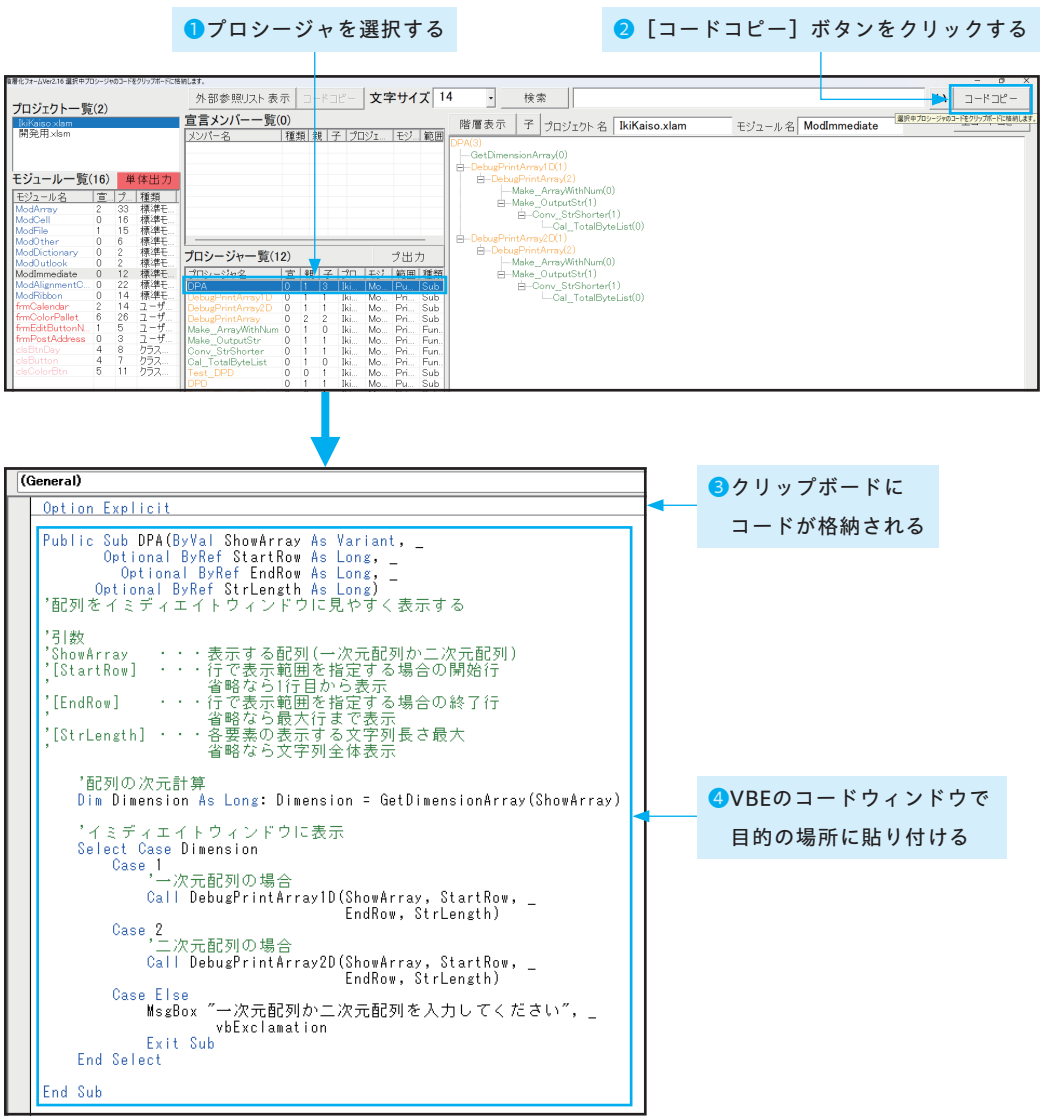
内容は子プロシージャを
すべて含めたもの

コメントで一覧表示される

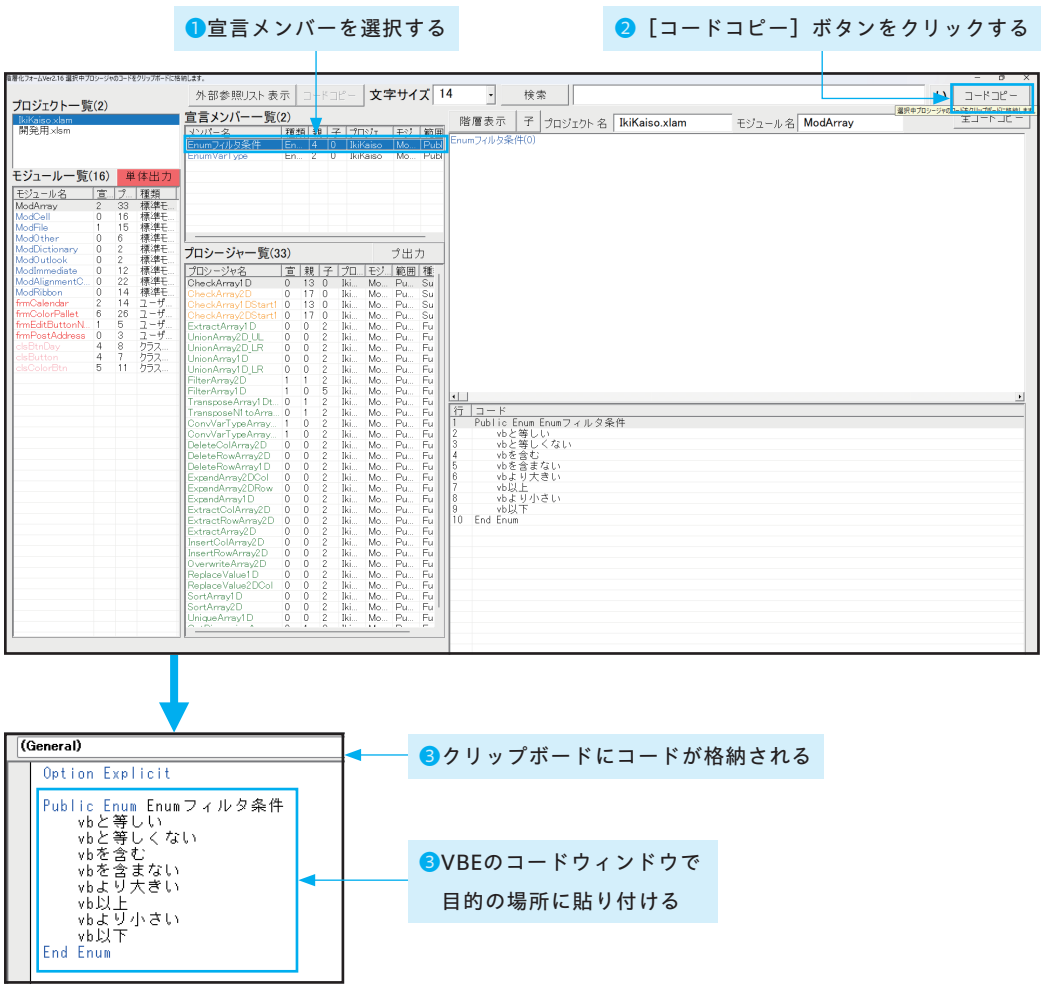


付 2 - 9 コードをクリップボードに格納する

画面右上の［コードコピー］ボタンで、プロシージャ単体のコードをクリップボードに格納することができます。

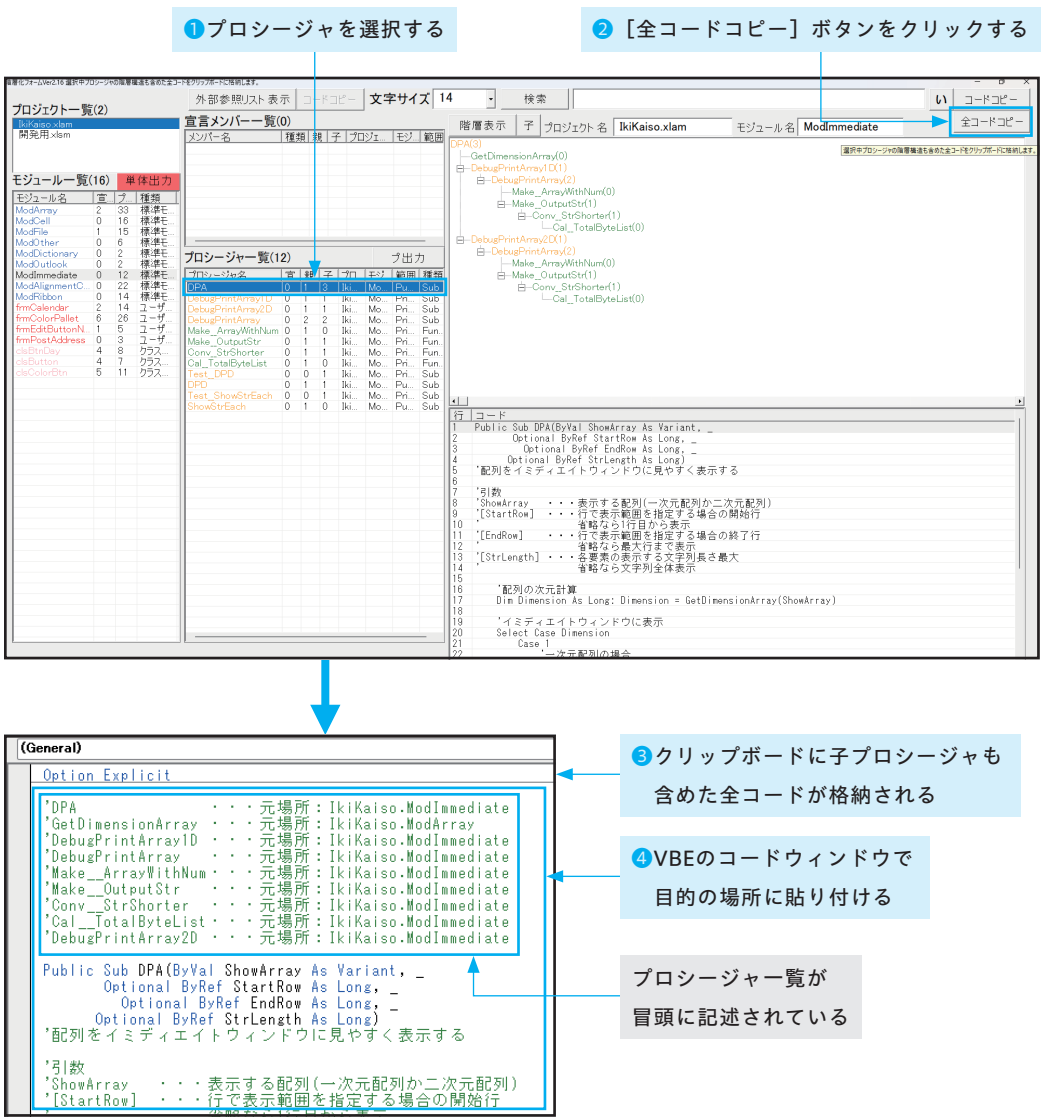


同様に、宣言メンバーの場合もコードをクリップボードに格納することができます。



付 2-10 プロシージャの全コードをクリップボードに格納する

画面右上の **[全コードコピー] ボタン**をクリックすると、プロシージャ一覧で選択したプロシージャとその子プロシージャまで含めた全コードをクリップボードに格納します。



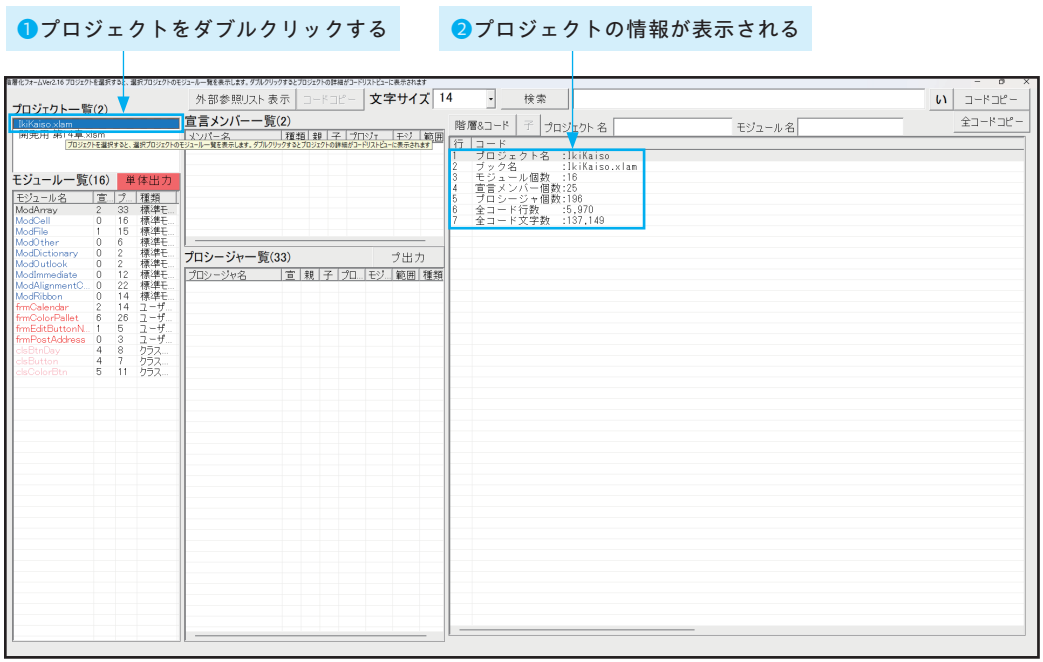
付 2-11 プロジェクト、モジュールの情報を表示する

プロジェクト一覧に表示されているプロジェクトをダブルクリックすると、コードにそのプロジェクトの情報が表示されます。

表示される情報は以下のとおりで、そのプロジェクトの規模などが確認できます。

- **プロジェクト名**
- **ブック名**
- **モジュール個数**
- **宣言メンバー個数**
- **プロシージャ個数**
- **全コード行数**
- **全コード文字数**

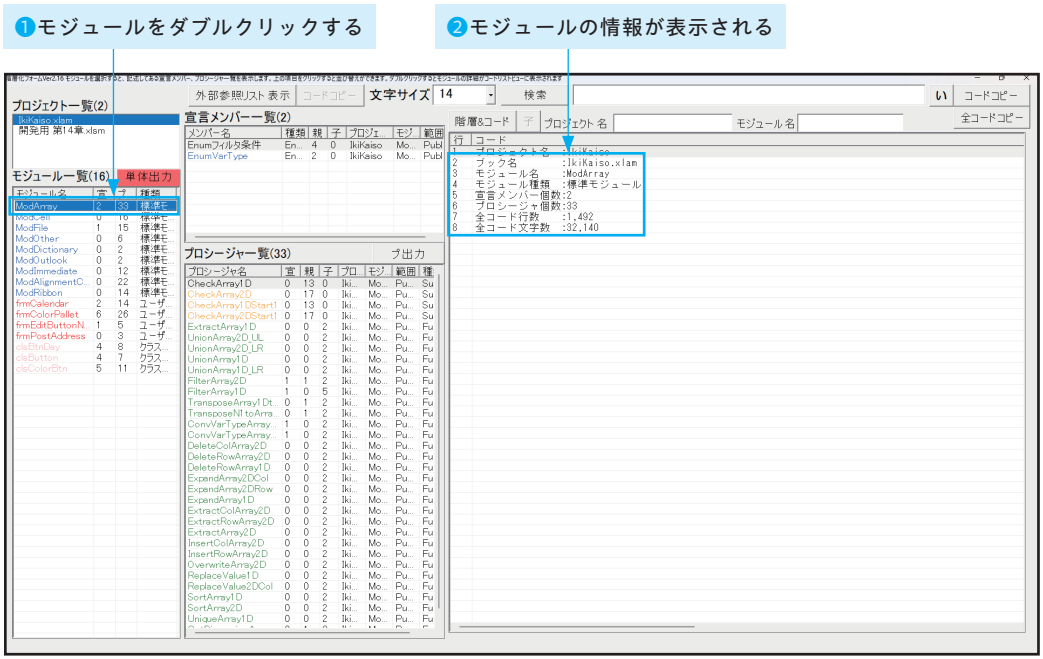
なお、全コード行数は空白行を除いた正味の行数で、全コード文字数もスペースは除いた正味の文字数になります。



同様に、モジュール一覧に表示されているモジュールをダブルクリックすると、そのモジュールの詳細情報がコードに表示されます。

表示される情報は以下のとおりで、モジュールの規模を確認できます。

- プロジェクト名
- ブック名
- モジュール名
- モジュール種類
- 宣言メンバー個数
- プロシージャ個数
- 全コード行数
- 全コード文字数



付 2-12 外部参照プロシージャの一覧を取得する

最後に紹介する「**外部参照プロシージャの一覧取得**」機能は、階層化フォームの一連の機能の中でも**もっとも役に立つ機能**です。

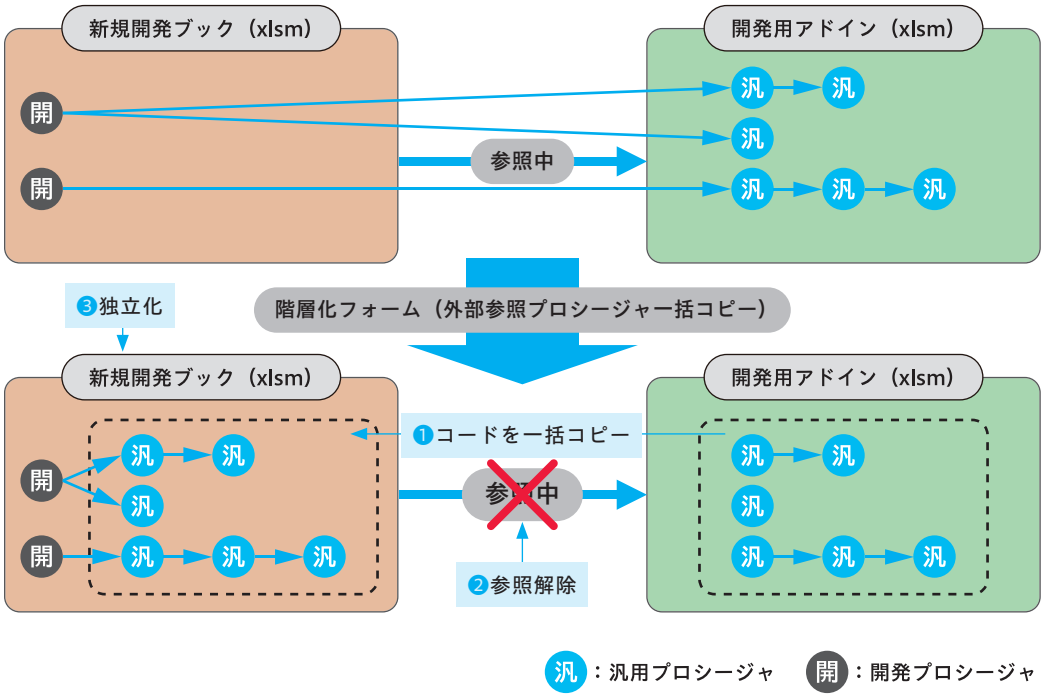
「外部参照プロシージャの一覧取得」とは、指定のプロジェクト（マクロ付きブック）で別のプロジェクトから参照しているプロシージャを一覧で取得する機能です。

たとえば、新規開発のマクロ付きブックが開発用アドインの汎用プロシージャを参照して開発されたものとします。すると、そのままでは新規開発のマクロ付きブックのマクロは開発用アドインを参照中でなければ動きません。

そこで、新規開発のマクロ付きブックのマクロが単独で動作するようにするためには、開発用アドイン内で使用している汎用プロシージャのコードをすべてマクロ付きブック内に複製する必要があります。

そして、この作業を一瞬で行える機能が「外部参照プロシージャの一覧取得」になります。

この機能は若干難解ですので、具体的な例を用いながら丁寧に解説します。

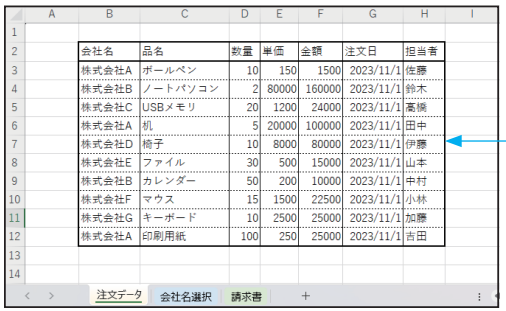


付2-12-1 新規マクロ付ブックを開発する

では、「外部参照プロシージャの一覧取得」機能の概要説明をしたところで、ここからは実例を用いて具体的かつわかりやすくこの機能について解説します。

まず、以下のような仕様のマクロ付きブックを開発します。

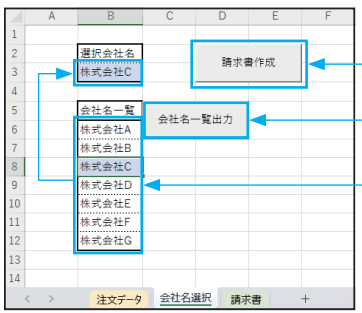
- **Workbook名は「請求書作成 体験サンプル.xlsm」とする**
- **Worksheetは「注文データ」「会社名選択」「請求書」の3つとする**
- **「注文データ」シートには注文データの表を貼り付ける**
- **「会社名選択」シートで「会社名一覧出力」ボタンを押すと「注文データ」をもとに重複のない会社名を一覧で出力する**
- **「会社名選択」シートで会社名一覧のセル範囲(セルB6以下)をダブルクリックすると「選択会社名」(セルB3)に反映される**
- **「会社名選択」シートで「請求書作成」ボタンを押すと選択会社名の請求書が「請求書」シートに出力される**
- **「請求書」シートにて「請求書PDF出力」ボタンを押すと「[会社名]_請求書.pdf」で請求書をPDFデータとして出力する**



会社名	品名	数量	単価	金額	注文日	担当者
株式会社A	ボールペン	10	150	1500	2023/11/1	佐藤
株式会社B	ノートパソコン	2	80000	160000	2023/11/1	鈴木
株式会社C	USBメモリ	20	1200	24000	2023/11/1	高橋
株式会社A	机	5	20000	100000	2023/11/1	田中
株式会社D	椅子	10	8000	80000	2023/11/1	伊藤
株式会社E	ファイル	30	500	15000	2023/11/1	山本
株式会社B	カレンダー	50	200	10000	2023/11/1	中村
株式会社F	マウス	15	1500	22500	2023/11/1	小林
株式会社G	キーボード	10	2500	25000	2023/11/1	加藤
株式会社A	印刷用紙	100	250	25000	2023/11/1	吉田

注文データ　会社名選択　請求書

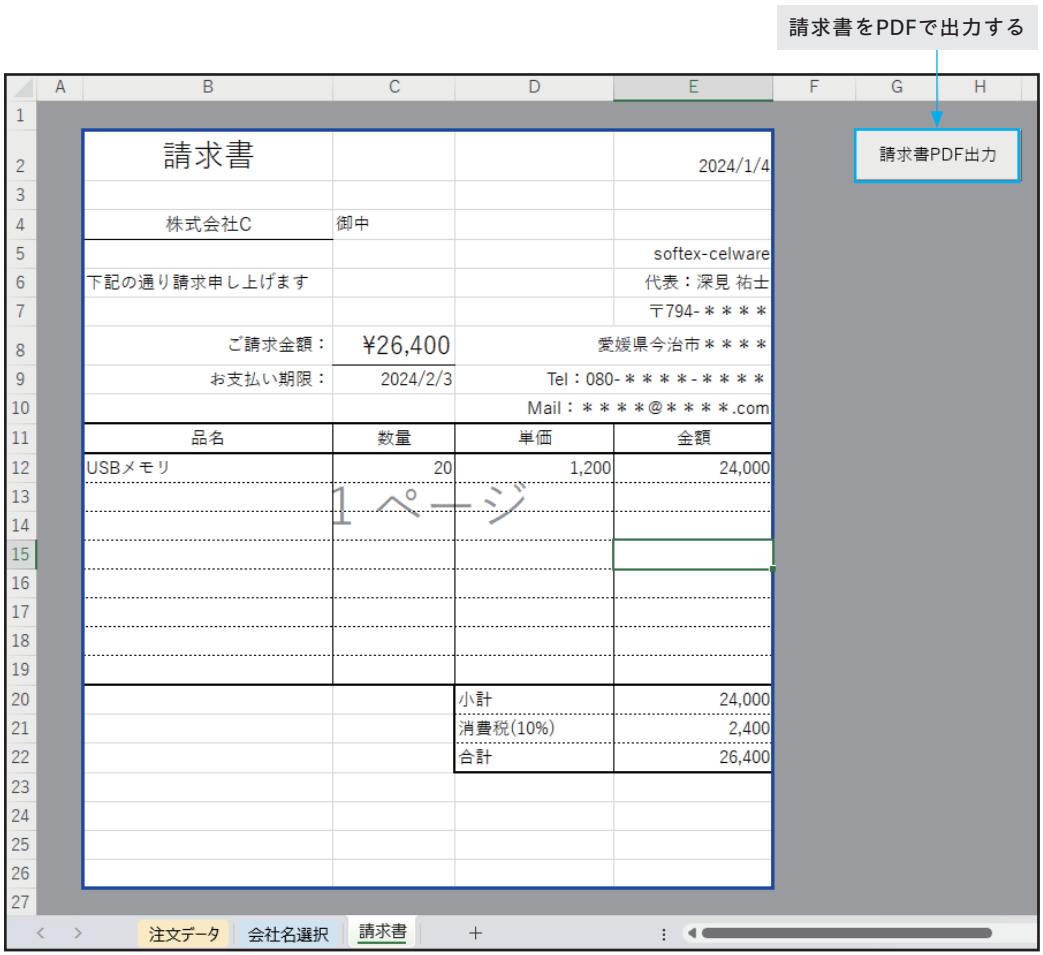
注文データを貼り付ける



選択会社名の請求書を作成する

重複しない会社名一覧を出力する

ダブルクリックして選択会社名を設定する



次に、実際に構築したVBAコードを説明します。

この時点では、開発用アドイン「IkiKaiso.xlam」の汎用プロシージャを参照している点に留意してください。

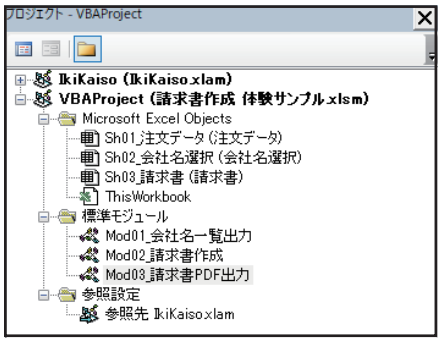
まず、3つのWorksheetのオブジェクトのオブジェクト名は、次のようにします。

- **「注文データ」シート → Sh01_注文データ**
- **「会社名選択」シート → Sh02_会社名選択**
- **「請求書」シート → Sh03_請求書**

そして、次の3つの標準モジュールを用意します。

- Mod01_会社名一覧出力
- Mod02_請求書作成
- Mod03_請求書PDF出力

以上のように設定すると、プロジェクトエクスプローラーは、次のようになります。



そして、各機能別のVBAコードを見ていきましょう。

●［会社名一覧出力］ボタン

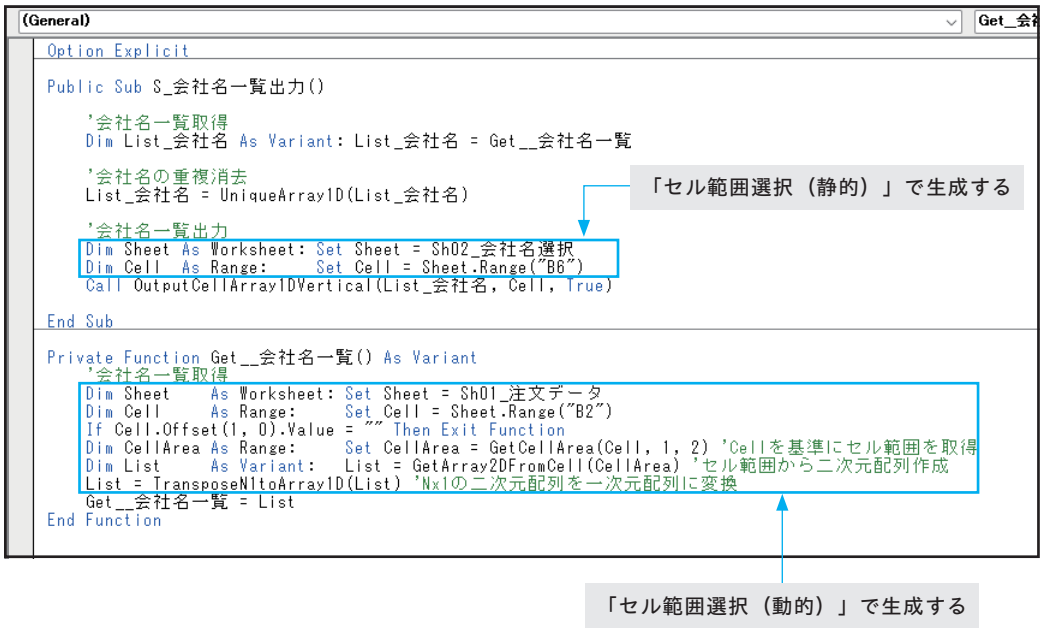
このボタンに登録されたマクロで、「注文データ」シートから会社名を一次元配列で取得して、重複を消去し、「会社名選択」シートに出力しています。

コードは標準モジュール「Mod01_会社名一覧出力」に記述しています。

なお、使用している汎用プロシージャは、次のとおりです。

- UniqueArray1D
- OutputCellArray1DVertical
- GetCellArea
- GetArray2DFromCell
- TransposeN1toArray1D

コーディングは、本書の第12章で紹介したリボン登録マクロ「セル範囲取得（動的）」「セル範囲取得（静的）」を利用して効率的に記述できます。



●ダブルクリックで選択会社名設定機能

「会社名選択」シートで出力済みの会社名一覧（「セルB6」以下）をダブルクリックすると、「セルB3」の選択会社名に設定できる機能です。

コードはWorksheetオブジェクト「Sh02_会社名選択」に記述しています。
使用している汎用プロシージャは、次のとおりです。

●GetCellArea

コーディングは、本書の第12章で紹介したリボン登録マクロ「セル範囲取得（動的）」を利用して効率的に記述できます。

```
(General)                                                    (Declara
Option Explicit

Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
'会社名一覧のセル範囲をダブルクリックして、選択会社名に反映
    Dim SelectCell As Range
    If Target.Count > 1 Then Set Target = Target(1)
    Set SelectCell = Get__会社名セル範囲 '選択範囲
    If Not Intersect(Target, SelectCell) Is Nothing Then
        Range("B3").Value = Target.Value '入力セル
        Cancel = True
    End If
End Sub

Private Function Get__会社名セル範囲() As Range
    Dim Sheet     As Worksheet: Set Sheet = Sh02_会社名選択
    Dim Cell      As Range:     Set Cell = Sheet.Range("B5")
    If Cell.Offset(1, 0).Value = "" Then Exit Function
    Dim CellArea As Range:      Set CellArea = GetCellArea(Cell, 1, 2) 'Cellを基準にセル範囲を取得
    Set Get__会社名セル範囲 = CellArea
End Function
```

「セル範囲選択（動的）」で生成して一部消去する

付録2
階層化フォームの使い方

●「請求書作成」ボタン

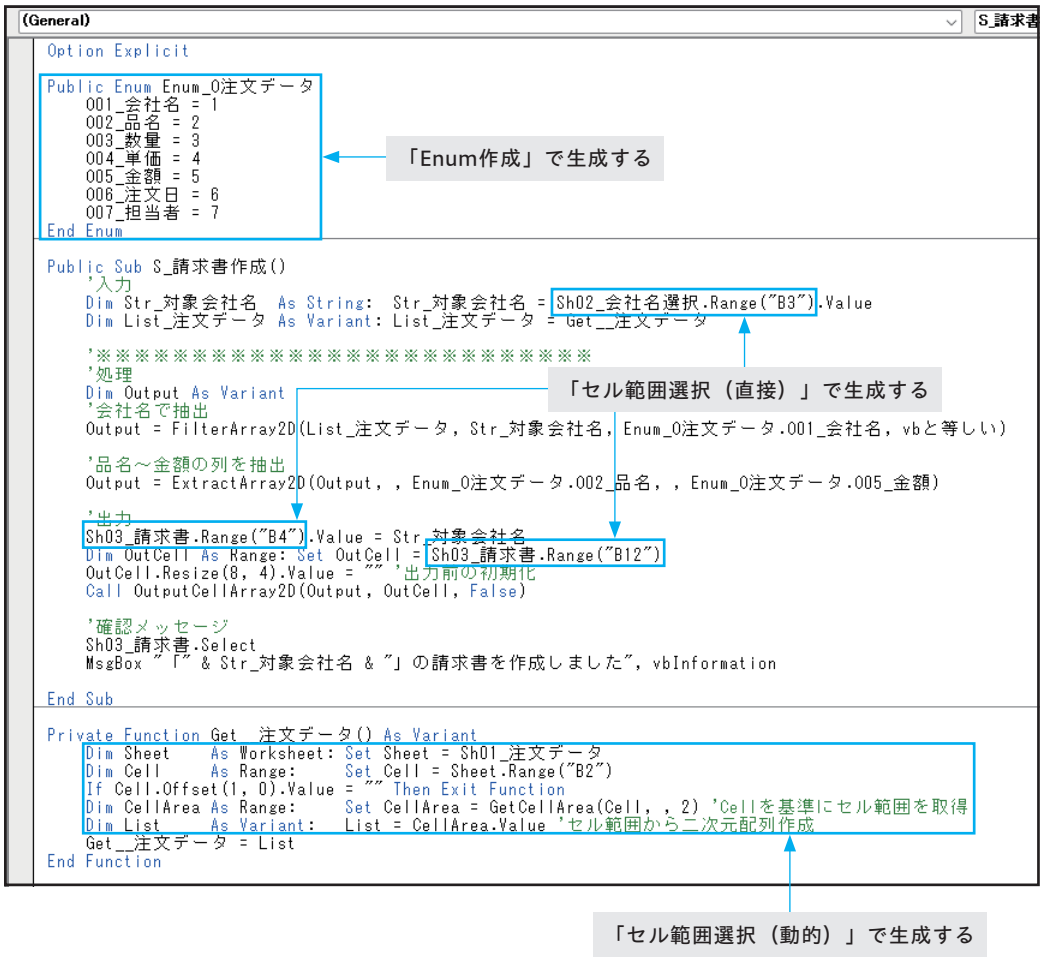
「会社名選択」シートで選択会社名に設定された会社名で請求書を「請求書」シートに出力する機能です。

「注文データ」シートから二次元配列として取得した注文データを、選択会社名で抽出して、必要な列だけを抽出したものを「請求書」シートの「セルB12」を基準に出力し、選択会社名を「セルB4」に出力して、最後に確認メッセージを表示します。

コードは標準モジュール「Mod02_請求書作成」に記述しています。
使用している汎用プロシージャは、次のとおりです。

- **FilterArray2D**
- **ExtractArray2D**
- **OutputCellArray2D**
- **GetCellArea**

コーディングは、本書の第12章で紹介したリボン登録マクロ「セル範囲取得（動的）」「セル範囲取得（直接）」「Enum作成」を利用して効率的に記述できます。



●「請求書PDF出力」ボタン

「請求書」シートに出力済みの請求書をPDFとして出力します。

出力先のフォルダは「ThisWorkbook.Path」と記述してマクロ実行中のブックと同じフォルダとし、PDFファイル名は会社名（セルB4）をもとに「[会社名]_請求書.pdf」とします。

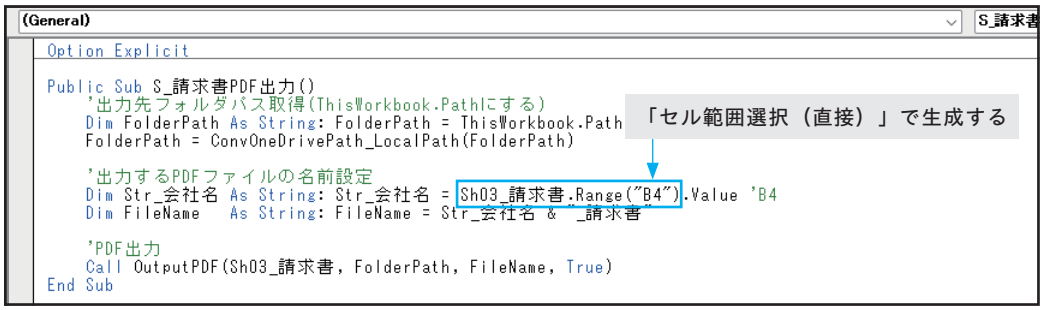
コードは標準モジュール「Mod03_請求書PDF出力」に記述しています。

使用している汎用プロシージャは、次のとおりです。

●ConvOneDrivePath_LocalPath

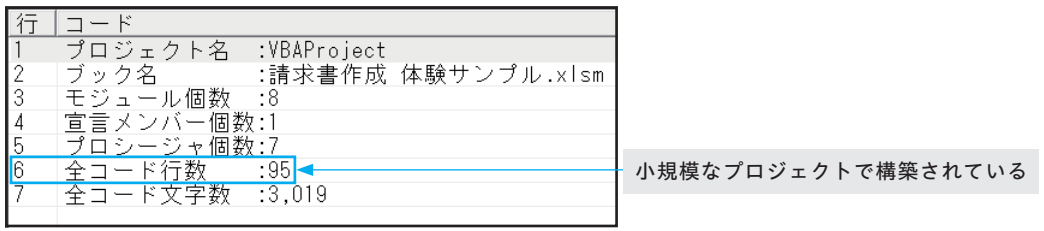
●OutputPDF

コーディングは、本書の第12章で紹介したリボン登録マクロ「セル範囲取得（直接）」を利用して効率的に記述できます。



ちなみに、上記仕様のプロジェクトの規模を階層化フォームで確認すると次のようになります。

合計4つの機能を実装しましたが、汎用プロシージャを利用しているのでコード行数は100行足らずの小規模で済んでいます。



付2-12-2 「外部参照プロシージャー覧取得」を使ってみる

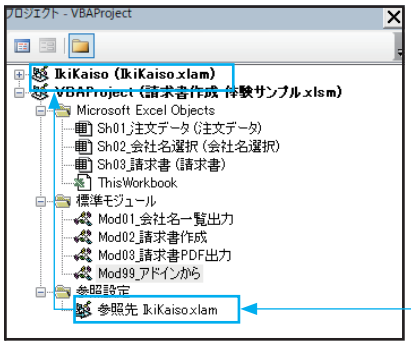
次に、階層化フォームの「外部参照プロシージャ一覧取得」を実際に使用してみます。

まず確認として、開発したマクロ付きブック「請求書作成 体験サンプル.xlam」は開発用アドイン「IkiKaiso.xlam」を参照して「IkiKaiso.xlam」の中の汎用プロシージャを利用しているため、単独では機能しません。

この状態のままだと、依頼者に提供するさいは「IkiKaiso.xlam」とセットで提供するか、「IkiKaiso.xlam」内で使用している汎用プロシージャのコードをすべて「請求書作成 体験サンプル.xlam」に複製する必要があります。

このとき、「IkiKaiso.xlam」とセットで提供するのは依頼者にとって使いづらい仕様になりますし、なによりも技術の流出につながってしまいます。

また、汎用プロシージャのコードをすべて複製するのは、参照している汎用プロシージャを1つずつ確認してコピー＆ペーストを繰り返すので大きな手間がかかります。



「請求書作成 体験サンプル.xlam」は
「IkiKaiso.xlam」を参照したままなので
単独では機能しない

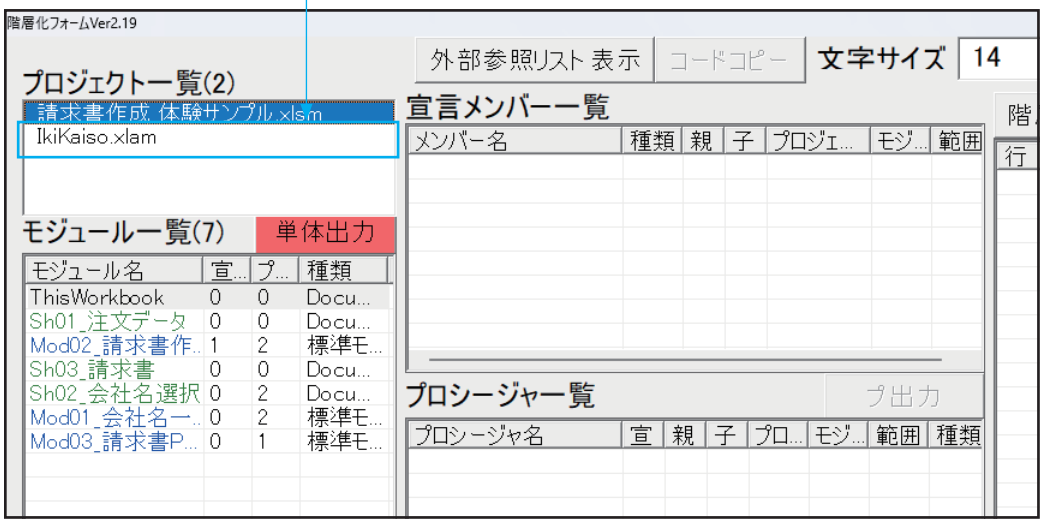
以上の説明でピンときた人もいると思いますが、「外部参照プロシージャ一覧取得」機能は、この「**汎用プロシージャのコードをすべて複製する**」作業を一瞬で行うものになります。

では、実際にやってみましょう。

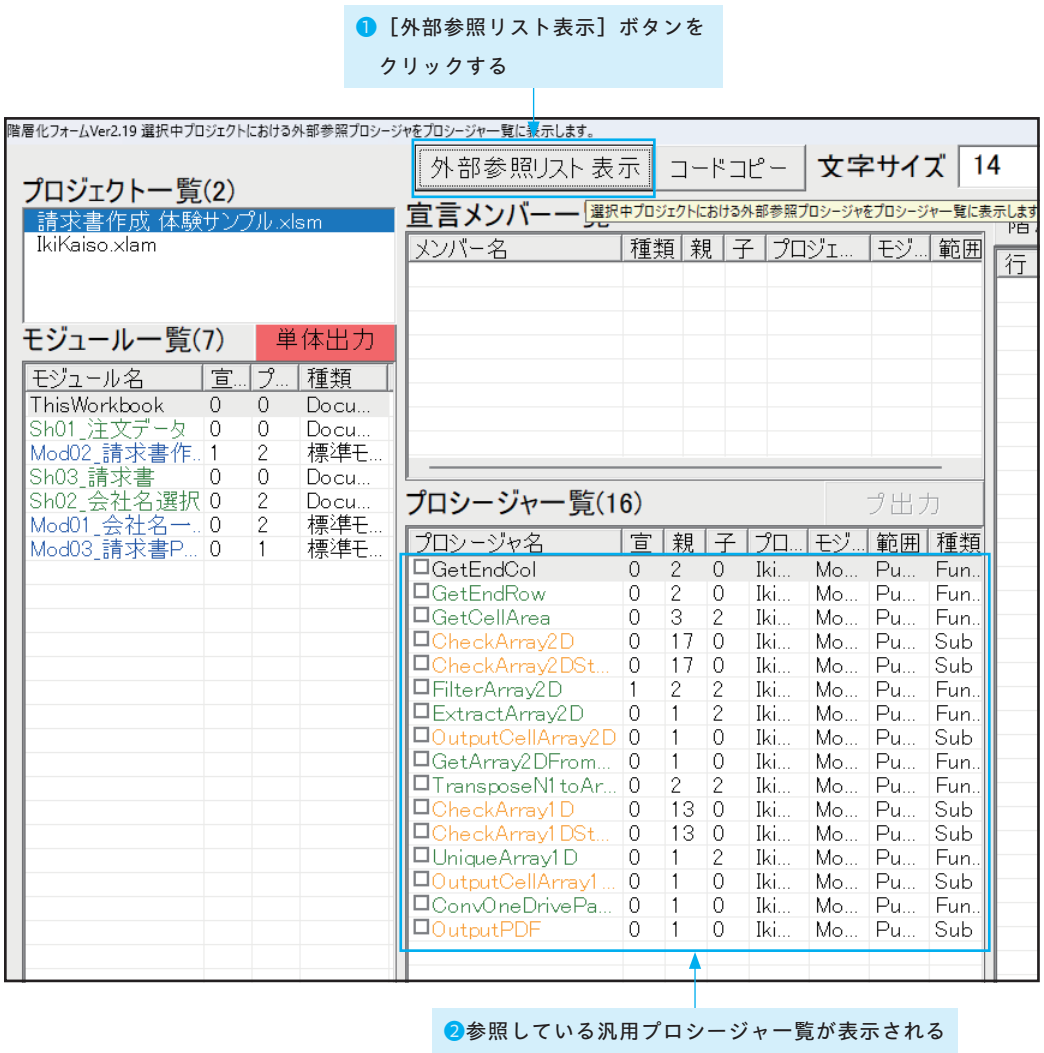
まずリボンの「階層化フォーム起動」で階層化フォームを起動します。

そして、プロジェクト一覧で「請求書作成 体験サンプル.xlam」を選択します。

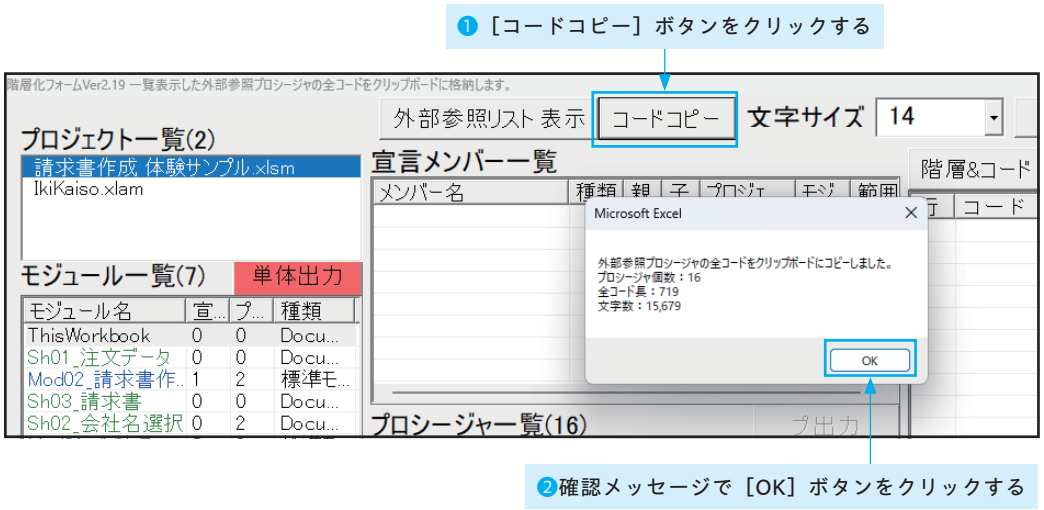
「請求書作成 体験サンプル.xlam」を選択する

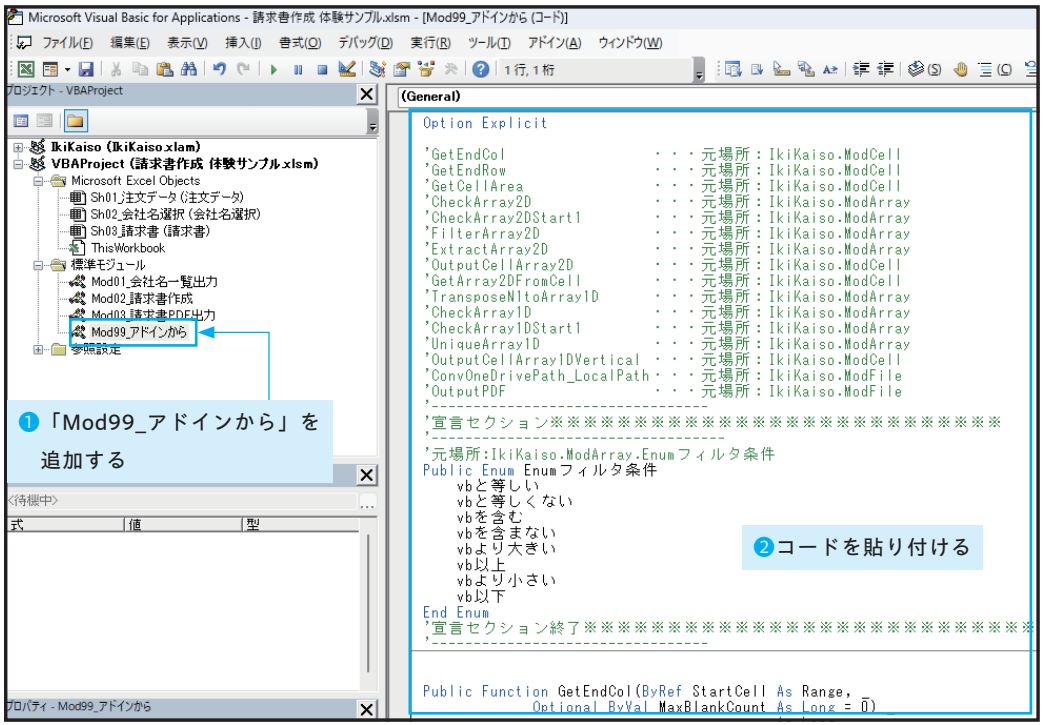


次に**[外部参照リスト表示] ボタン**を押すと、プロシージャ一覧で参照している汎用プロシージャが一覧で表示されます。



そして、［外部参照リスト表示］ボタンの右にある**[コードコピー] ボタン**をクリックすると、表示している参照中の汎用プロシージャ一覧のコードがまとめてクリップボードに格納されます。





❶［コードコピー］ボタンをクリックする

モジュール名	宣...	プ...	種類
ThisWorkbook	0	0	Docu...
Sh01_注文データ	0	0	Docu...
Mod02_請求書作..	1	2	標準モ...
Sh03_請求書	0	0	Docu...
Sh02_会社名選択	0	2	Docu...

外部参照プロシージャの全コードをクリップボードにコピーしました。
プロシージャ個数：16
全コード長：719
文字数：15,679

❷確認メッセージで［OK］ボタンをクリックする

次に、「請求書作成　体験サンプル.xlam」に標準モジュール「Mod99_アドインから」を新規追加して、その中にクリップボードに格納されているコードを貼り付けます。

❶「Mod99_アドインから」を追加する

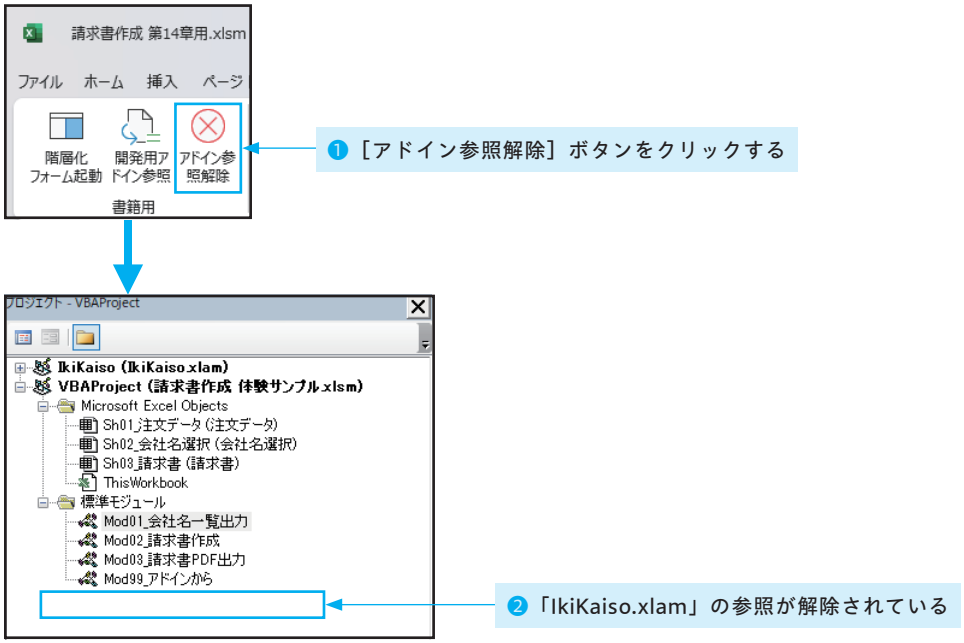
```
Option Explicit

'GetEndCol                      ・・・元場所：IkiKaiso.ModCell
'GetEndRow                      ・・・元場所：IkiKaiso.ModCell
'GetCellArea                    ・・・元場所：IkiKaiso.ModCell
'CheckArray2D                   ・・・元場所：IkiKaiso.ModArray
'CheckArray2DStart1             ・・・元場所：IkiKaiso.ModArray
'FilterArray2D                  ・・・元場所：IkiKaiso.ModArray
'ExtractArray2D                 ・・・元場所：IkiKaiso.ModArray
'OutputCellArray2D              ・・・元場所：IkiKaiso.ModCell
'GetArray2DFromCell             ・・・元場所：IkiKaiso.ModCell
'TransposeN1toArray1D           ・・・元場所：IkiKaiso.ModArray
'CheckArray1D                   ・・・元場所：IkiKaiso.ModArray
'CheckArray1DStart1             ・・・元場所：IkiKaiso.ModArray
'UniqueArray1D                  ・・・元場所：IkiKaiso.ModArray
'OutputCellArray1DVertical      ・・・元場所：IkiKaiso.ModCell
'ConvOneDrivePath_LocalPath     ・・・元場所：IkiKaiso.ModFile
'OutputPDF                      ・・・元場所：IkiKaiso.ModFile
'---------------------------------
'宣言セクション※※※※※※※※※※※※※※※※※※※※※※※※※※※
'-----------------------------------
'元場所:IkiKaiso.ModArray.Enumフィルタ条件
Public Enum Enumフィルタ条件
    vbと等しい
    vbと等しくない
    vbを含む
    vbを含まない
    vbより大きい
    vb以上
    vbより小さい
    vb以下
End Enum
'宣言セクション終了※※※※※※※※※※※※※※※※※※※※※※※※※※※
'---------------------------------


Public Function GetEndCol(ByRef StartCell As Range, _
             Optional ByVal MaxBlankCount As Long = 0)
```

❷コードを貼り付ける

最後に、［アドイン参照解除］ボタンで開発用アドイン「IkiKaiso.xlam」の参照を解除すれば、「請求書作成 体験サンプル.xlam」は単独で動くマクロ付きブックになります。



もちろん、最後の動作確認は必要ですが、これで依頼者に安心してマクロ付きブックを単独で提供できます。

ちなみに、外部参照プロシージャを「Mod99_アドインから」に複製後、再び階層化フォームで「請求書作成 体験サンプル.xlam」のプロジェクト規模を確認すると次のようになります。

行	コード
1	プロジェクト名　:VBAProject
2	ブック名　　:請求書作成 体験サンプル.xlsm
3	モジュール個数　:8
4	宣言メンバー個数:2
5	プロシージャ個数:23
6	全コード行数　　:776
7	全コード文字数　:17,771

全コード行数が95行から776行に8.1倍増えている

外部参照プロシージャの複製前、すなわち「**実際にコーディングした行数**」が95行でしたが、複製後は776行なので、681行分は汎用プロシージャで流用できたことが確認できます。

割合で見ると約8割は汎用プロシージャの流用なので、汎用プロシージャを使用せずに最初からすべて記述した場合と比較すると「**約8倍の効率化**」が実現できています。

さらにリボンに登録したマクロ「セル範囲選択」や「Enum作成」も利用してレコードを自動生成もしているので、**実質の効率は10倍**くらいでしょう。

以上、説明は大変長いものになってしまいましたが、これが「**外部参照プロシージャー覧取得**」機能になります。

いかがですか。この機能を利用すれば、開発用アドインに構築した汎用プロシージャを100%利用して、VBA開発が目を疑うレベルで効率化できるのが実感できたのではないでしょうか。

そして最後になりますが、汎用プロシージャの構築は第7章までで説明したルールを参考にしっかりと整理整頓し、管理をする必要がありますが、今後**どれだけたくさんの汎用プロシージャを増やしても、「階層化フォーム」があれば新規開発ブックにおいて簡単に流用ができます。**

すなわち、**汎用プロシージャを増やせば増やすほど、加速度的にみなさんのVBA開発は効率化されていく**ことになるのです。

付録2
階層化フォームの使い方