Visual COBOL チュートリアル

COBOL 開発：Visual Studio – 静的コード解析機能

1. 目的

本チュートリアルでは、静的コード解析ツールの利用方法の習得を目的としています。

静的コード解析ツールは、プログラムに内在する下記のような問題を検出することができます。

- 潜在バグになりえるようなコーディング

  未初期化変数の利用、動作が規定されない命令の利用
- 性能の観点上、非効率なコーディング

  ゾーン十進変数を使った算術演算、リトルエンディアン環境におけるビックエンディアン変数の使用
- ソースの可読性・保守性を低下させるようなコーディング

  デッドコード、GO TO 句の多用

一般的な開発では、コーディングルールの中で上記のような問題となる記述をしないように規約として定義します。Visual COBOL は、これらの規約をルールとして定義し、定義に基づいた問題点の検出を実施できます。Visual COBOL ではビルドインルールが利用できますが、弊社別製品である Enterprise Analyzer と連携することで、独自のカスタムルールを追加することができるようになります。

なお、本機能は、ネイティブ COBOL のみに有効です。また、 .NET COBOL プロジェクトに追加されたネイティブ COBOL プログラムには利用できません。

2. 前提

- 本チュートリアルで使用したマシン OS ： Windows 11
- Visual COBOL 9.0 for Visual Studio がインストール済みであること

下記のリンクから事前にチュートリアル用のサンプルファイルをダウンロードして、任意のフォルダに解凍しておいてください。

サンプルプログラムのダウンロード

内容

3. チュートリアル手順の概要

本チュートリアルでは、いくつかのルールに違反したコードを内在する短いプログラムを利用して機能の確認を行います。

3.1. IDE からの実行

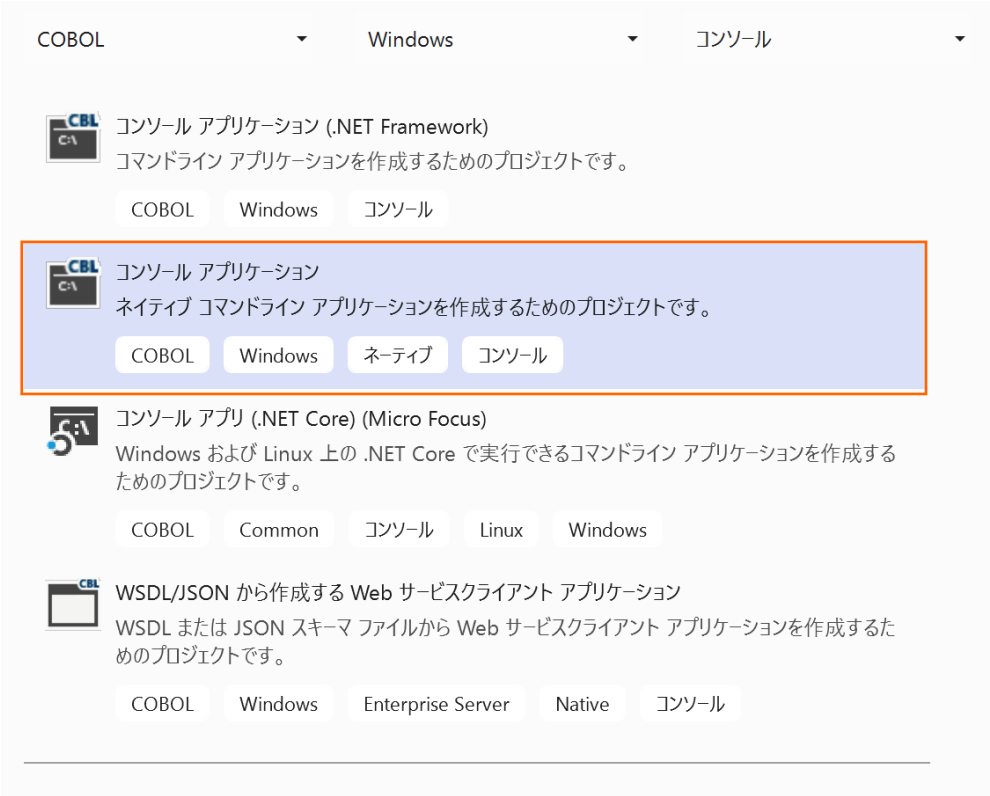
3.1.1. Visual Studio の起動

1) スタートメニューより、Visual Studio 2022 を起動します。

3.1.2. 静的コード解析の実施

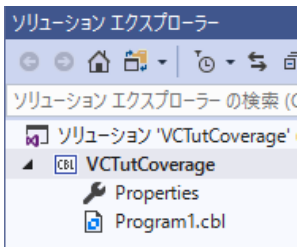
1) [新しいプロジェクトの作成] をクリックします。

2) 言語リストに"COBOL"、プロジェクトタイプに"コンソール" を選択し、「コンソール アプリケーション」を選択した上で、[次へ(N)] ボタンをクリックします。



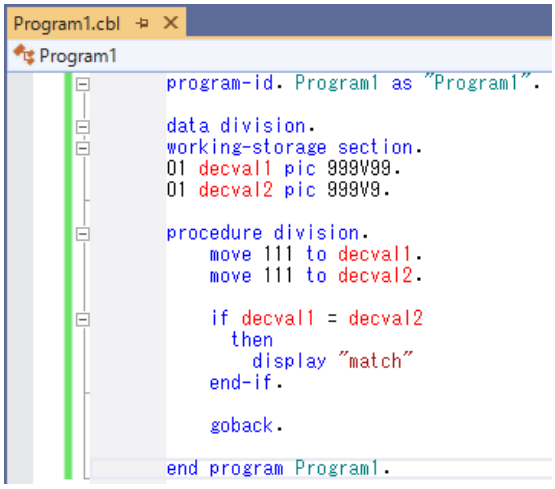
3) プロジェクト名に "VCTutCodeAnalysis" を入力し、[作成(C)] ボタンをクリックします。

新規プロジェクトが作成されます。



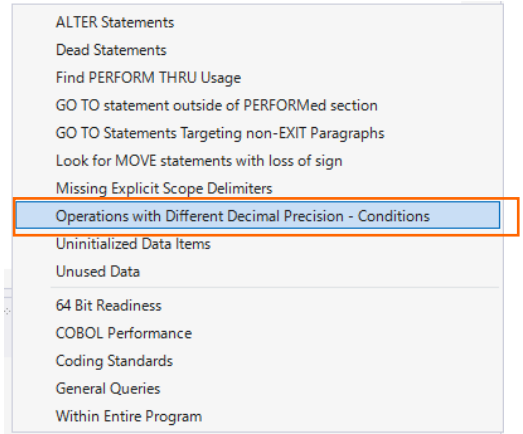
4) サンプルプログラム Program1.cbl でコードを上書き保存します。

これは、静的コード解析ルールに違反するコードを含めた簡単なプログラムとなります。



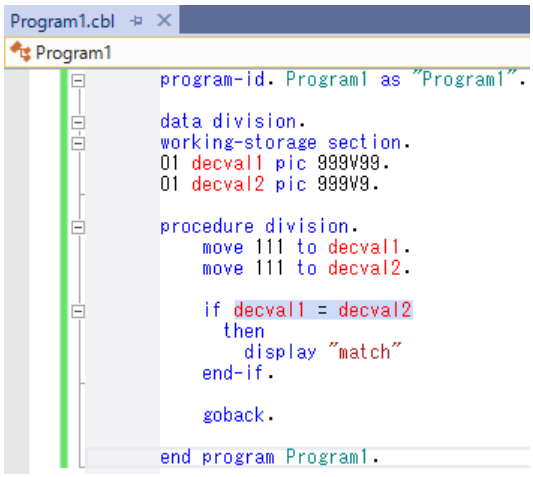
5) VCTutCodeAnalysis プロジェクト名を選択した状態で、マウスの右クリックにてコンテクストメニューを表示し、[Code Analysis] > [Operations with Different Decimal Precision – Conditions] を選択します。

このルールは、小数点位置が異なる数値を比較している個所を検出します。

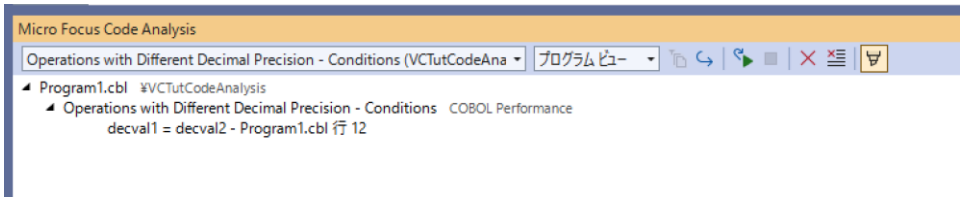


補足）
本チュートリアルでは、プロジェクト名を選択しているため、プロジェクトに含まれる全ての COBOL プログラムが静的コード解析対象となっています。任意の COBOL プログラム名を選択して実行することで、解析対象を限定することができます。

6) コード解析でルールに違反している個所が、エディタ上でハイライト表示されます。

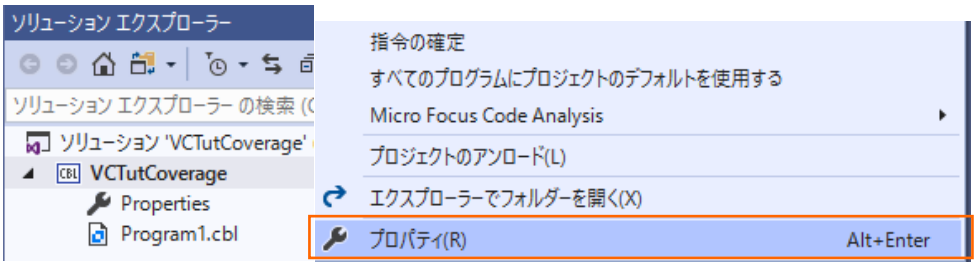


また、Micro Focus Code Analysis ビューでは結果の一覧を確認することができます。

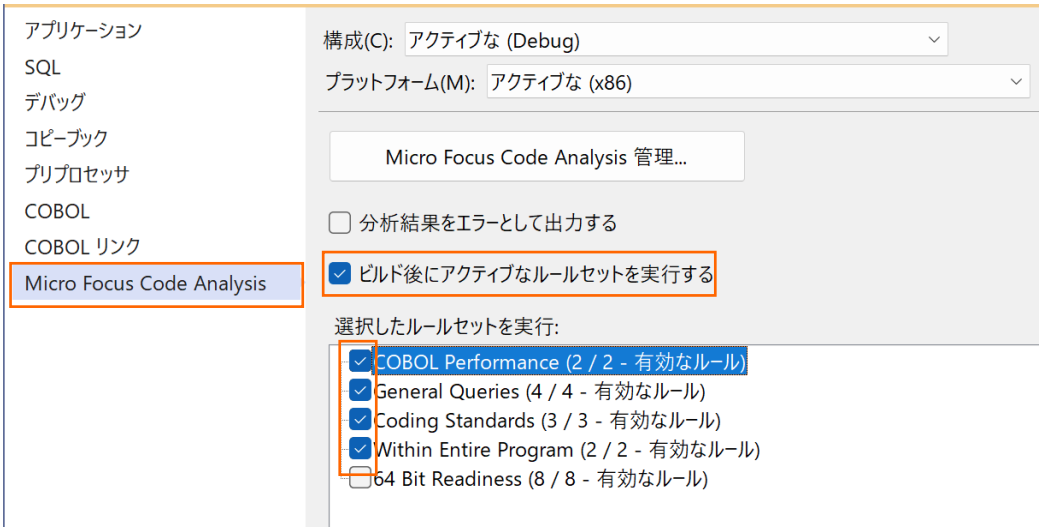


3.1.3. プログラムビルド時の自動実行

1) VCTutCodeAnalysis プロジェクト名を選択した状態で、マウスの右クリックにてコンテクストメニューを表示し、[プロパティ(R)] ボタンをクリックします。

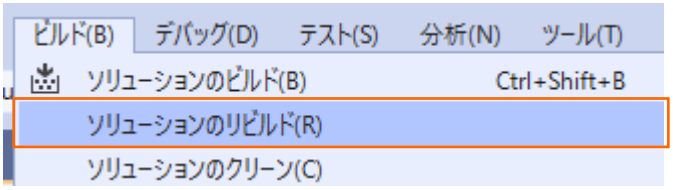


2) Micro Focus Code Analysis タブを選択し、「ビルド後にアクティブなルールセットを実行する」にチェックを行い、アクティブにするルールセットを選択した上で、変更を保存します。

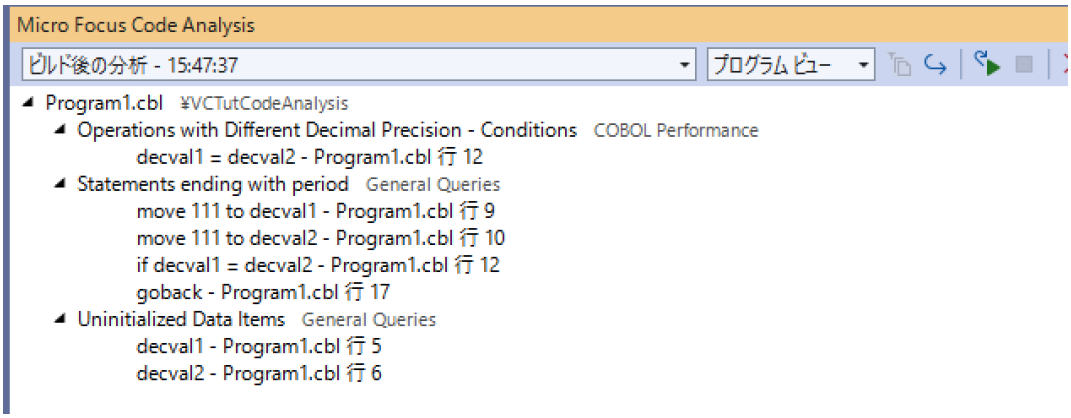


3)

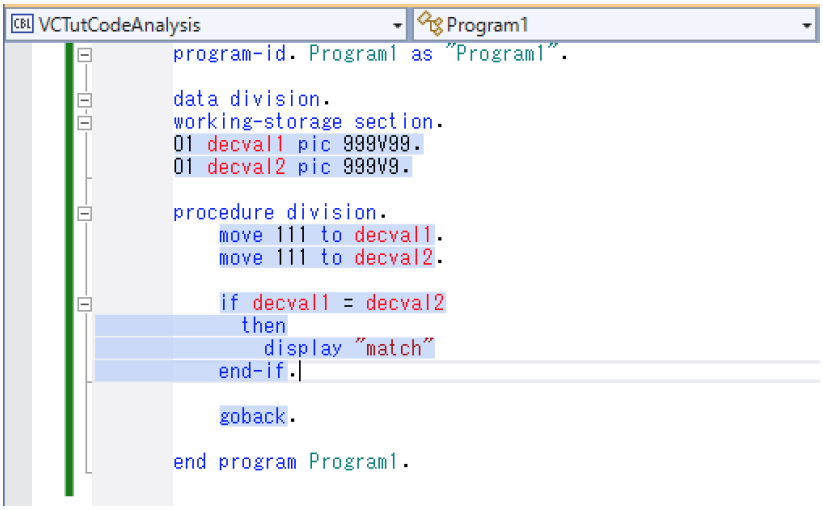
4) Visual Studio IDE メニューより、[ビルド(B)] > [ソリューションのリビルド(R)] を選択します。



Micro Focus Code Analysis ビューより、前手順で確認した際と異なるルール違反が検出されていることを確認します。



エディタ上にも、各違反箇所がハイライトされます。



3.2. カスタムルールの追加

弊社別製品である Enterprise Analyzer をルールエディタとして利用することで独自ルールの作成することができ、作成されたルールを Visual COBOL にインポートすることで実現できます。本手順については、Enterprise Analyzer 製品マニュアルを参照ください。

3.3. ビルドインされたルール一覧

3.3.1. Operations with Different Decimal Precision – Conditions

小数情報の精度が異なる変数に対する操作を検出するルールとなります。

```
01 decval1 pic 999V99.
01 decval2 pic 999V9.

 procedure division.
*> (中略)
    if decval1 = decval2
    then
        display "match"
    end-if.
```

3.3.2. Find PERFORM THRU Usage

PERFORM THRU 句を用いて、複数回同じ section 句の実行を検出するルールとなります。

```
procedure division.
    perform para1 thru para3.
    display "between performs".
    perform para2 thru para3.
    goback.

para1.
    display "para1".

para2.
    display "para2".

para3.
    display "para3".
```

3.3.3. GO TO Statements Targeting non-EXIT Paragraphs

EXIT 句が存在しない段落に対して GO TO 句を使用した遷移を検出するルールとなります。

```
procedure division.
    go to sec2.
    goback.

sec1.
    display "sec1".
    goback.

sec2.
    display "sec2".

end program Program4.
```

3.3.4. Uninitialized Data Items

値の初期化をすることなく、プログラム内で使用している変数を検出するルールとなります。

```
data division.
working-storage section.
01 val pic x(6).

procedure division.
    initialize val.
    display val.
    goback.
```

3.3.5. GO TO Statement outside of PERFORMed section

PERFORM 句により実行された Section 節内で、Section 外への遷移を検出するルールとなります。

```
procedure division.
* GO TO statement outside of PERFORMed section
    perform sec1.
    goback.

sec1 section.
    go to para1.

sec2 section.

para1.

end program
```

3.3.6. Look for MOVE statements with loss of sign

符号なしの数値定義に対し、符号つきの値を設定するため、符号情報が欠落する箇所検出するルールとなります。

```
data division.
working-storage section.
01 val pic 9(4).

procedure division.
    move -123 to val.
```

3.3.7. Missing Explicit Scope Terminators

範囲符が未記載のため、暗黙的にスコープが終了している箇所を検出するルールとなります。その他範囲符については、Visual COBOL の製品マニュアルから以下のページを参照ください。

[リファレンス] > [COBOL 言語リファレンス] > [第 1 部： 言語の概念] > [COBOL 言語の概念] > [データの字類および項類] > [明示指定および暗示指定] > [明示範囲符および暗示範囲符]

```
        procedure division.

      * Explicit Scope Terminators
            if 1 = 2
            then
                display "1 won't match with 2"
            else
                display " not match"
            .
```

3.3.8. ALTER Statements

ALTER 句の利用により、動的にプログラムフローを書き換えている箇所を検出するルールとなります。

```
        procedure division.

        alter main to proceed to sec2
        perform main.
        goback.
        main.
        display "tgt".
        go to sec1.

        sec1.
            display "sec1".

        sec2.
            display "sec2".
```

3.3.9. Dead Statements

プログラムの記述上、実行されない不要な記述を検出するルールとなります。

```
        procedure division.
            goback.
            display "DEAD CODE".
```

3.3.10. Unused Data

定義はあるものの、プログラム内で使用していない変数を検出するルールとなります。

```
data division.
working-storage section.
01 unused pic x.
01 used pic x.
procedure division.
    move "X" to used.
    goback.
```

3.3.11. Statements ending with period

命令の最後がピリオドで終わっていないステートメントを検出するルールとなります。

```
data division.
working-storage section.
01 work pic x(3).
procedure division.
    Move “AAA” to work.
    DISPLAY work
```

3.4. 代表的なルールセット

ルールセットとは、複数の静的コード解析ルールをまとめたもので、セット内のルールを一括で解析実行することができます。それぞれ、以下のビルドインルールが含まれています。

3.4.1. COBOL Performance

- Operations with Different Decimal Precision – Conditions
- Find PERFORM THRU Usage

3.4.2. General Queries

- GO TO Statements Targeting non-EXIT Paragraphs
- Uninitialized Data Items
- GO TO Statement outside of PERFORMed section
- Statements ending with period

3.4.3. Coding Standards

- Look for MOVE statements with loss of sign

- Missing Explicit Scope Terminators
- ALTER Statements

3.4.4. Within Entire Program

- Dead Statements
- Unused Data

WHAT'S NEXT

- 本チュートリアルで学習した技術の詳細については製品マニュアルをご参照ください。

免責事項