

Visual COBOL チュートリアル

COBOL 開発：ステップバイステップチュートリアル

Visual Studio 2022

1 目的

Visual COBOL for Visual Studio 2022 は、マイクロソフト社の最新開発環境である Visual Studio 2022 の IDE（統合開発環境）上で COBOL のアプリケーション開発を行うための製品です。COBOL プログラマーが既存の COBOL 資産を Windows 環境で活用するだけでなく、COBOL 言語のプログラミング経験のないプログラマーが初めて COBOL アプリケーション開発を行う場合に最適な製品です。

このドキュメントは、Visual COBOL for Visual Studio 2022 を学ぶためのステップバイステップのチュートリアルです。

2 前提

- 本チュートリアルで使用したマシン OS : Windows 11
- Visual COBOL 11.0 for Visual Studio がすでにインストールされていること
インストール手順は以下の FAQ サイトを参照してください。
https://support.amc.rocketsoftware.co.jp/SupportInf/amc_faqlpublic.aspx?VC01002
- プログラミングの基礎知識を有していること
- Windows の基本操作を理解していること

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](#)

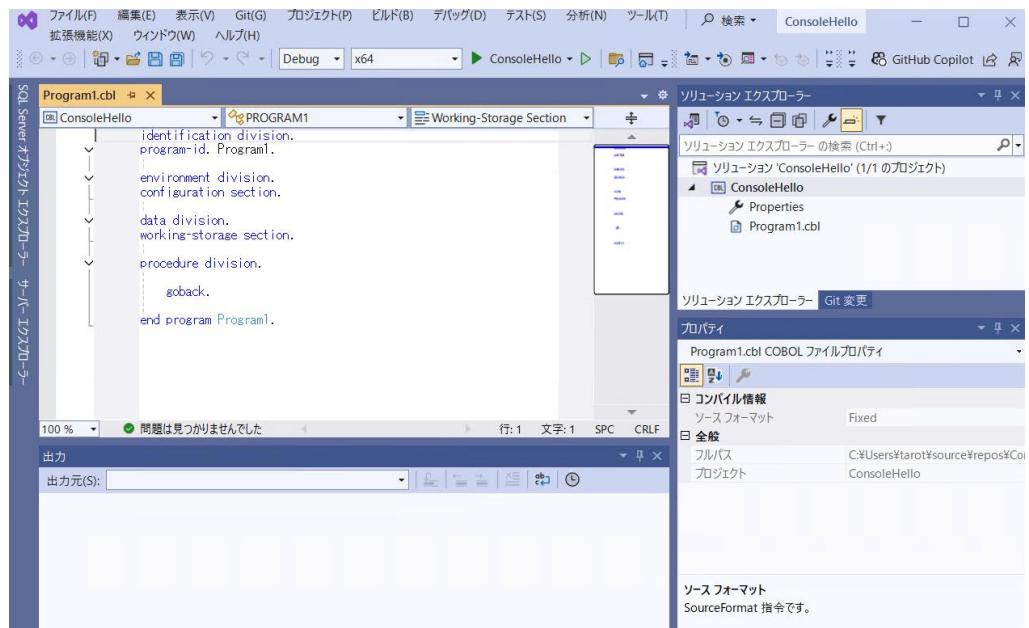
内容

- 1 目的
- 2 前提
- 3 チュートリアル
 - 3.1 Visual Studio の IDE に慣れてみよう
 - 3.2 はじめての Visual COBOL プロジェクト
 - 3.3 ファイルの入出力

3 チュートリアル

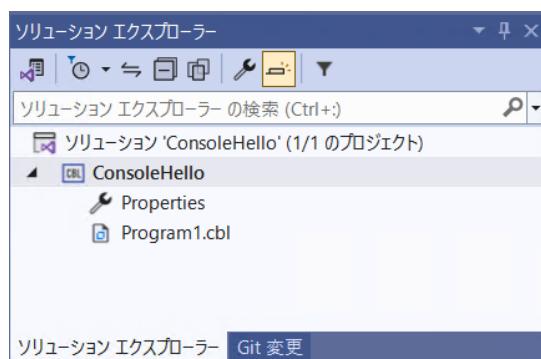
3.1 Visual Studio の IDE に慣れてみよう

- 1) Windows のスタートメニューから Visual Studio 2022 をクリックします。
- 2) Visual Studio を起動して任意の COBOL プロジェクトを作成すると以下のような画面が表示されます。



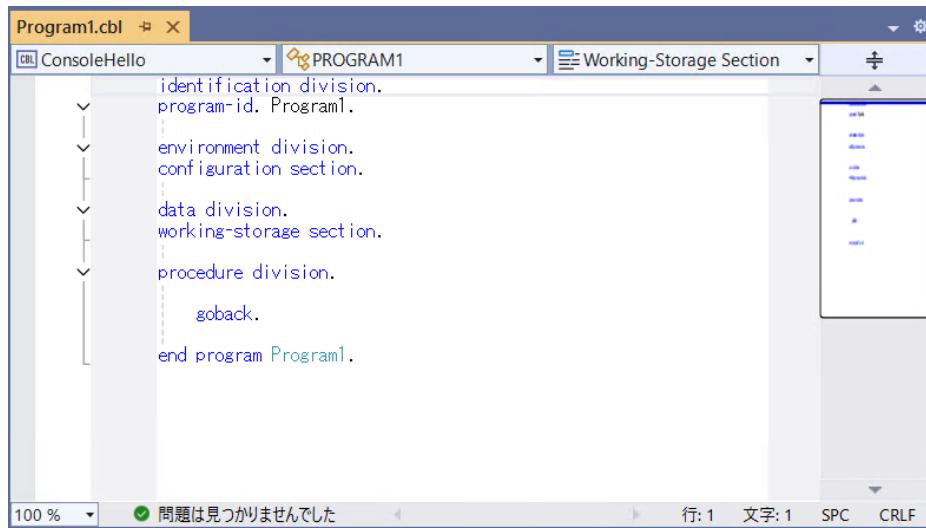
Microsoft Visual Studio 2022 の IDE は、メニューバー、ツールバー、左、下または右にドッキングまたは自動的に非表示になる各種ツールウィンドウ、エディター領域など、複数の要素で構成されます。IDE 内の要素の配置は、適用した設定とその後に加えたカスタマイズ内容によって異なります。

Visual Studio 2022 のソリューションとプロジェクトには、アプリケーションの作成に必要な参照、データ接続、フォルダ、およびファイルを表す項目が含まれています。ソリューションには複数のプロジェクトを含めることができます。プロジェクトには、通常、複数の項目が含まれます。ソリューションエクスプローラーには、ソリューション、それらのプロジェクト、そのプロジェクト内の項目が表示されます。ソリューションエクスプローラーを使用すると、編集するファイルを開く、プロジェクトに新規ファイルを追加する、ソリューション、プロジェクト、および項目のプロパティを表示するなどの操作を実行できます。



Visual Studio 2022 のソースコードエディターには、COBOL 予約語とデータ名や手続き名などの利用者語を色分け表示や、COBOL スニペットなど COBOL 言語固有の機能拡張が含まれます。ソースコードを入力するとバックグラウンドチ

ツイックを実行して、赤の波線でエラー箇所を強調表示します。そのエラー箇所にマウスポインタを移動することでエラー内容の確認、定義への移動、他の参照検索などの操作が可能です。



```

Program1.cbl ✘ X
CBL ConsoleHello PROGRAM1 Working-Storage Section
identification division.
program-id. Program1.

environment division.
configuration section.

data division.
working-storage section.

procedure division.

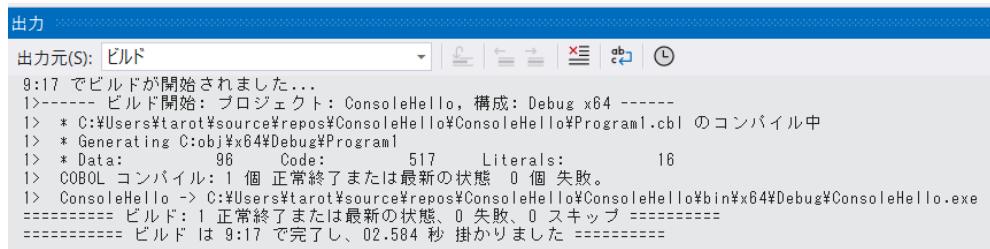
    goback.

end program Program1.

```

100 % 問題は見つかりませんでした 行: 1 文字: 1 SPC CRLF

Visual Studio 2022 のビルド構成では、プラットフォームの選択、プロジェクトまたはソリューションのビルド方法を定義します。プロジェクトタイプごとに、デバッグとリリースのデフォルト構成があり、独自の構成を作成することも可能です。コンソールウィンドウにはビルト時のメッセージやアプリケーションのコンソール出力等が表示されます。問題ウィンドウには、不正な構文、キーワードのスペルミス、型の不一致などのコンパイルエラーが表示されます。

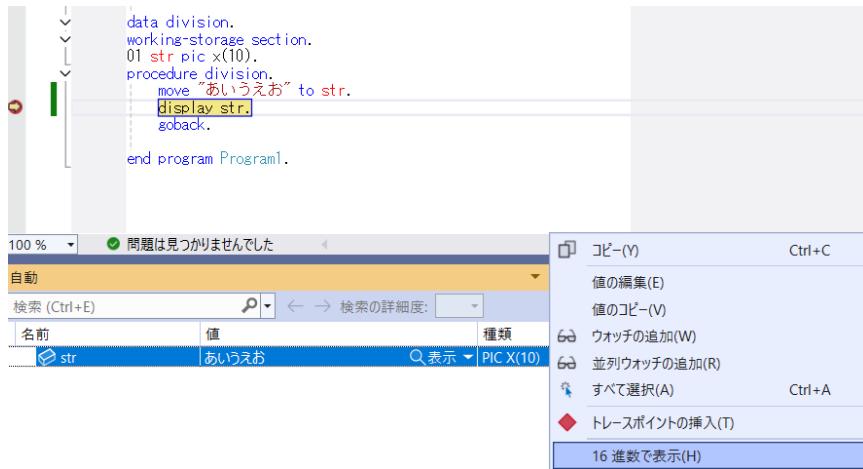


```

出力
出力元(S): ビルド
9:17 でビルトが開始されました...
1>----- ビルド開始: プロジェクト: ConsoleHello, 構成: Debug x64 -----
1> * C:\Users\tarot\source\repos\ConsoleHello\ConsoleHello\Program1.cbl のコンパイル中
1> * Generating C:\obj\x64\Debug\Program1
1> * Data: 98 Code: 517 Literals: 18
1> COBOL コンパイル: 1 個 正常終了または最新の状態 0 個 失敗。
1> ConsoleHello -> C:\Users\tarot\source\repos\ConsoleHello\bin\x64\Debug\ConsoleHello.exe
===== ビルド: 1 正常終了または最新の状態、0 失敗、0 スキップ =====
===== ビルドは 9:17 で完了し、02.584 秒 掛かりました =====

```

ビルトしたアプリケーションは、実行時の論理エラーやセマンティックエラーなどの問題を検出して修正するために、デバッガーを使用します。Visual Studio 2022 のデバッガーは、コードのステップ実行、様々な条件を設定したブレークポイントで実行、変数ウィンドウやウォッチ式などのツールを使用してローカル変数やその他の関連データを調べることができます。



```

data division.
working-storage section.
01 str pic x(10).
procedure division.
move "あいうえお" to str.
display str.
goback.

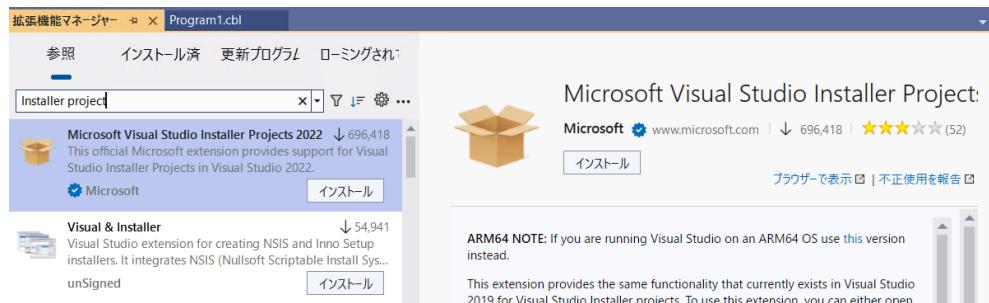
end program Program1.

```

100 % 問題は見つかりませんでした 自動 検索 (Ctrl+E) 検索の詳細度: 名前 値 種類 str あいうえお 表示 PIC X(10)

□ コピー(Y) Ctrl+C
値の編集(E)
値のコピー(V)
⌚ ウォッチの追加(W)
⌚ 並列ウォッチの追加(R)
すべて選択(A) Ctrl+A
◆ トレースポイントの挿入(T)
16進数で表示(H)

デバッグが完了したアプリケーションは、Windows インストーラーを使用するか、ファイルを手動でコピーして、本番環境に配置します。Visual Studio 2022 では、Visual Studio の Marketplace より「Microsoft Visual Studio Installer Projects」をダウンロードし、インストールすることで下図のような各種インストーラー作成用のプロジェクトをご利用できます。なお、本番環境には COBOL Server が事前にインストールされている必要があります。



3.2 はじめての Visual COBOL プロジェクト

- 1) Visual Studio 2022 を起動します。

Windows のスタートメニューから Visual Studio 2022 をクリックします。Microsoft Visual Studio 2022 が起動されるので「新しいプロジェクトの作成」を選択します。

フィルター画面が表示されるので、言語に“COBOL”、プラットフォームに“Windows”、プロジェクト タイプに“ネーティブ”を選択し、一覧から「コンソールアプリケーション」を選んで、[次へ(N)] ボタンをクリックします。



- 2) 以下の入力を行い、[作成(C)] ボタンをクリックします。

プロジェクト名： ConsoleHello

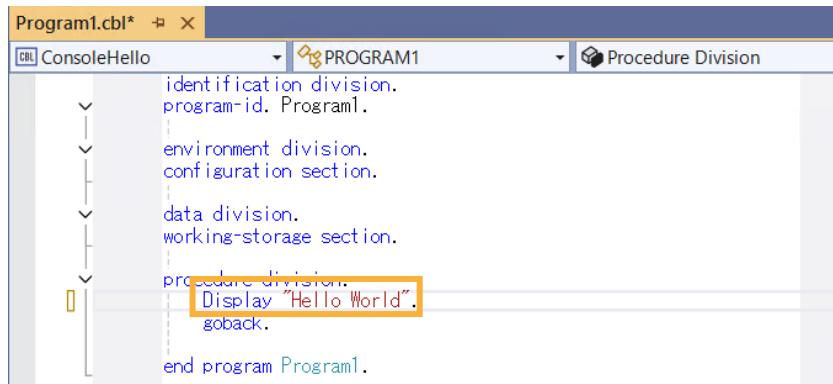
ソリューション名： TutorialSol



3) Visual Studio のコードエディターで COBOL のソースコードを編集します。

ConsoleHello プロジェクトの作成が成功すると、COBOL 専用のコードエディターが起動します。エディター画面には、コンソールアプリケーションのひな形が表示されています。COBOL ソースは、見出し部 (identification division)、環境部 (environment division)、データ部 (data division)、手続き部 (procedure division) で構成されますが、今回は「Hello World」を表示して終了するプログラムなので、手続き部に DISPLAY 文を書き加えるだけです。なお、COBOL 正書法ではエディター画面左右にあるグレー部分を特別な領域として利用するので、通常のソースコードはこれを避けて入力します。

今回は、procedure division の下に「Display "Hello World".」とタイプします。



```
Program1.cbl*  PROG1  Procedure Division
ConsoleHello  PROGRAM1
identification division.
program-id. Program1.

environment division.
configuration section.

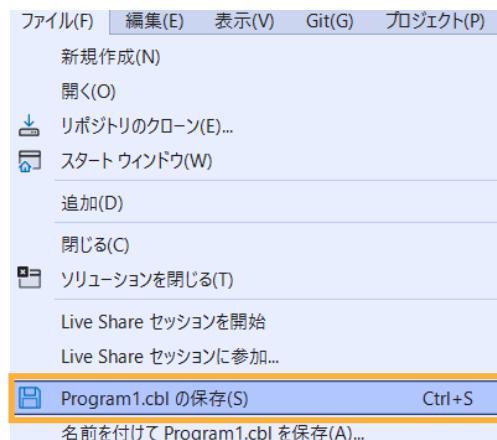
data division.
working-storage section.

procedure division.
Display "Hello World".
goback.

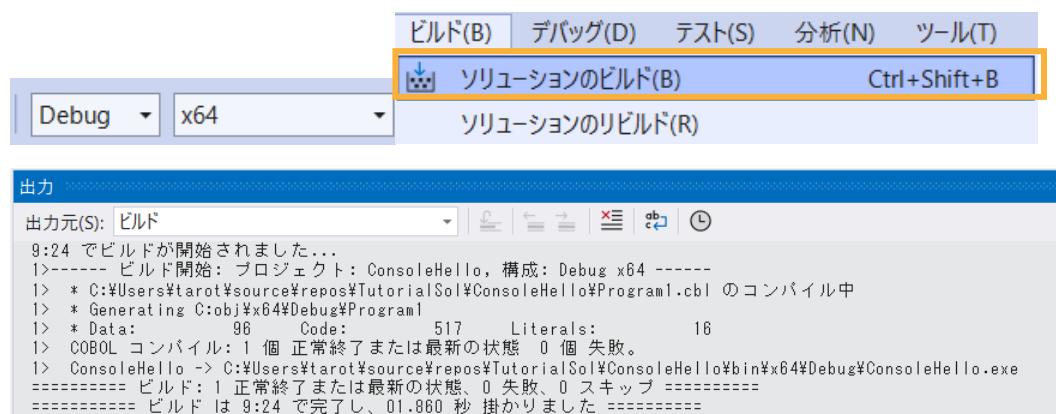
end program Program1.
```

4) COBOL アプリケーションをビルドします。

- ① 終止符 (ピリオド) を含めてスペルミスがなければ、メニューより、[ファイル(F)] > [Program1.cbl の保存] を選択します。

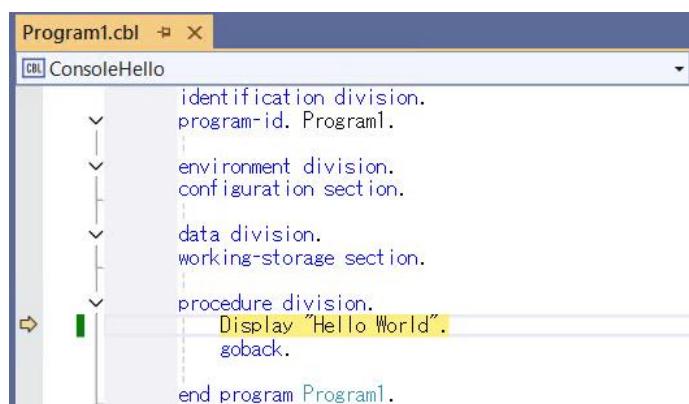


- ② ソリューション構成が Debug、ソリューションプラットフォームが x64 であることを確認して、メニューより、[ビルド(B)] > [ソリューションのビルド(B)] を選択します。出力ウィンドウにビルド結果が表示されるので、ビルドが正常終了したことを確認します。

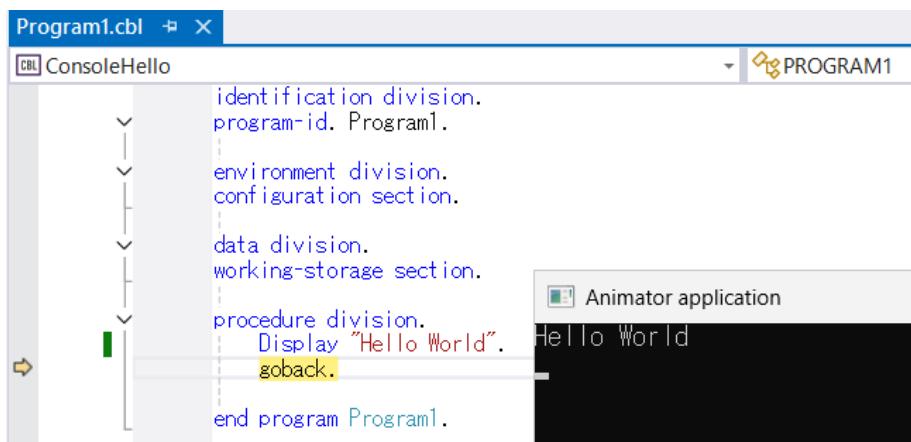


5) COBOL アプリケーションをデバッグ実行します。

メニューより、[デバッグ(D)] > [ステップイン(I)] を選択すると、コマンドプロンプト画面が開き、デバッガーがステップ実行を開始します。デバッガーは手続き部の最初の COBOL 文である display 文を実行する前の状態で停止します。今回は調べるローカル変数がないので、そのまま [ステップイン(I)] を選択し、ステップ実行を進めます。



ステップイン後の状態、プロンプト画面は最小化されていることがあります。



```

Program1.cbl ✘ X
CBL ConsoleHello PROGRAM1
identification division.
program-id. Program1.

environment division.
configuration section.

data division.
working-storage section.

procedure division.
Display "Hello World".
goback.

end program Program1.

```

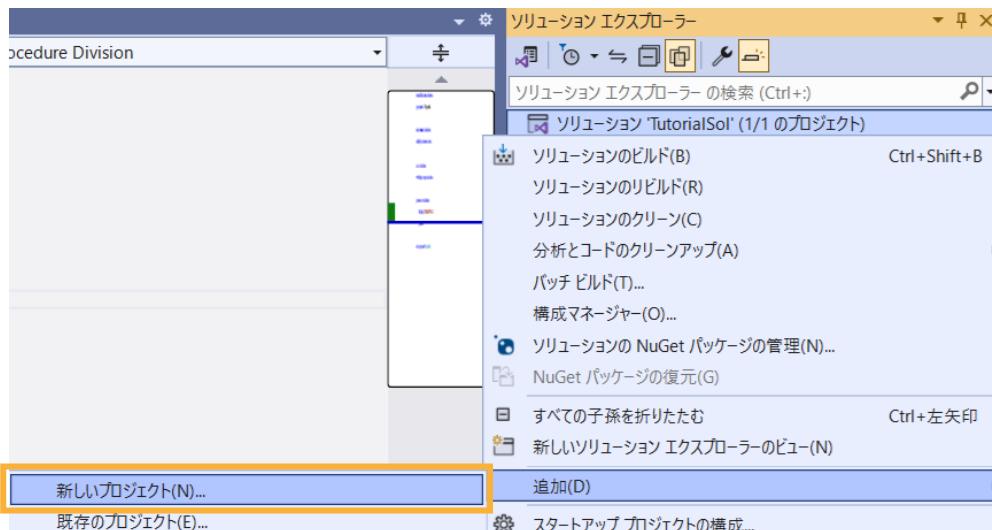
The screenshot shows the COBOL IDE interface. On the left is the code editor with the file 'Program1.cbl'. The code is a simple COBOL program that displays 'Hello World' and then goes back. On the right, there is a 'PROGRAM1' window showing the output of the program: 'Hello World'.

コマンドプロンプト画面に“Hello World”が表示されたことを確認して、デバッグを終了します。

続いて、ウィンドウ画面のボタンを押して“Hello World”を表示する COBOL アプリケーションを作成します。

6) 作成したソリューションへプロジェクトを追加します。

これまでに作成したソリューション中のソリューションエクスプローラーにて、ソリューションを選択した状態で、マウスの右クリックにてコンテキストメニューを表示し、[追加(D)] > [新しいプロジェクト(N)…] を選択します。



7) 使用するテンプレートを選択します。

フィルター画面が表示されるので、言語に“COBOL”、プラットフォームに“Windows”、プロジェクト タイプに“デスクトップ”を選択し、一覧から「Windows フォームアプリケーション(.NET Framework)」を選んで、[次へ(N)] ボタンをクリックします。

COBOL Windows デスクトップ

WCF サービス ライブラリ (.NET Framework)
WCF サービス クラス ライブラリを作成するためのプロジェクトです。
新規

COBOL Windows デスクトップ ライブラリ

配信サービス ライブラリ (.NET Framework)
WCF サービスとして公開される配信サービスです。
新規

COBOL Windows デスクトップ

Windows フォーム コントロール ライブラリ (.NET Framework)
Windows フォーム アプリケーションで使用するコントロールを作成するプロジェクトです。
新規

COBOL Windows デスクトップ ライブラリ

Windows フォーム アプリケーション (.NET Framework)
Windows フォーム、ユーザー インタフェースのあるアプリケーションを作成するためのプロジェクトです。
新規

COBOL Windows デスクトップ

空のプロジェクト (.NET Framework)
ローカル アプリケーションを作成するための空のプロジェクトです。
新規

COBOL Windows デスクトップ

ユニット テスト ライブラリ (.NET Framework)
MUnit テスト ライブラリを作成するための .NET プロジェクトです。

次へ(N)

- 8) プロジェクト名に “WinHello” を入力し、[作成(C)] ボタンをクリックします。

Windows フォーム アプリケーション (.NET Framework) COBOL Windows デスクトップ

プロジェクト名(I)
WinHello

場所(L)
C:\Users\tarot\source\repos\TutorialSol

フレームワーク(F)
.NET Framework 4.7.2

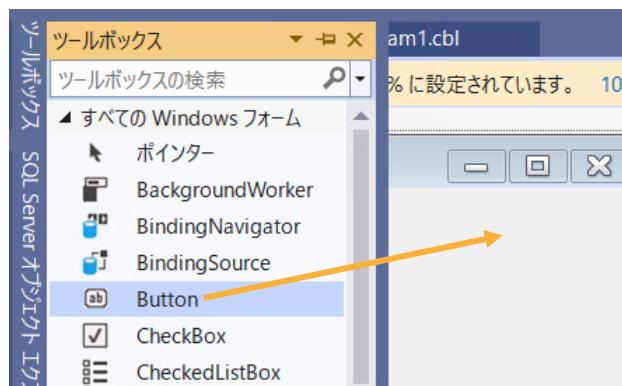
プロジェクトは “C:\Users\tarot\source\repos\TutorialSol\WinHello” で作成されます

戻る(B) 作成(C)

9) フォームデザイナーでウィンドウを作成します。

WinHello プロジェクトの作成が成功すると、フォームデザイナーが起動します。

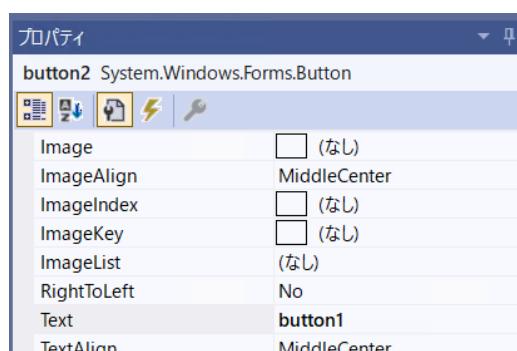
デザイナー画面に Form1 ウィンドウが表示された後、[表示(V)] > [ツールボックス(X)] を選択します。表示されたツールボックス中のすべての Windows フォームを展開します。続いて、Button コントロールを選択し、Form1 ウィンドウ上にドラッグ&ドロップします。



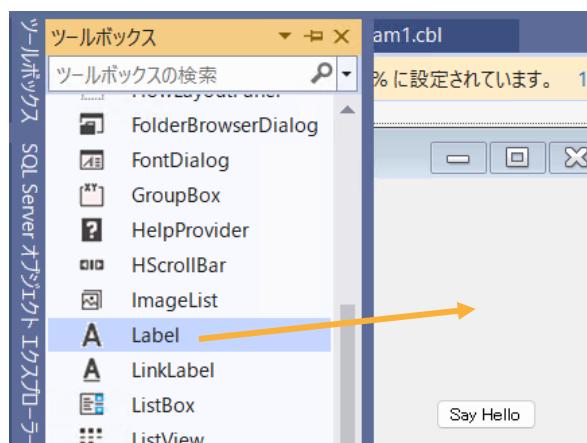
Form1 ウィンドウ上にボタンが表示されると、プロパティが Button1 ボタンに切り替わります。プロパティを下方向にスクロールして「Text」を選択し、値を "Say Hello" に変更します。

(補足)

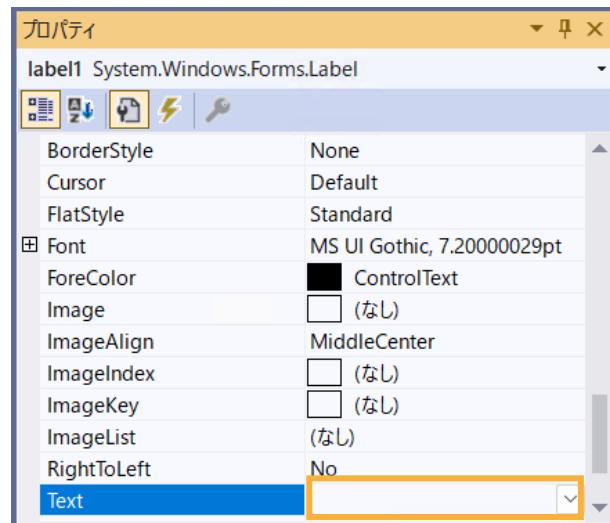
プロパティは、通常、右下に表示されます。表示されていない場合は、Button1 を選択した状態で、マウスの右クリックにてコンテキストメニューを開き、[プロパティ] を選択してください。



ツールボックスをスクロールして Label コントロールを選択し、Form1 ウィンドウ上にドラッグ&ドロップします。



プロパティをスクロールして「Text」を選択し、テキストの値を削除します。

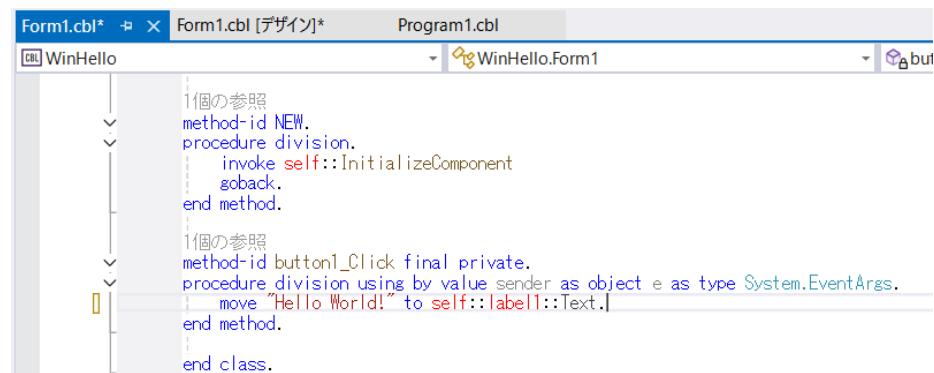


10) コードエディターで COBOL ソースコードを入力します。

デザイナー画面上の [Say Hello] ボタンをダブルクリックすると、COBOL 専用のコードエディターが起動します。

エディター画面には、Windows フォームアプリケーションのひな形が表示されます。ここでは [Say Hello] ボタンをクリックした時の処理を記述するので、Button1_Click メソッドの手続き部に以下の move 文を追加します。

move "Hello World!" to self::label1::Text.



```

Form1.cbl* ✎ ✎ Form1.cbl [デザイン]* Program1.cbl
[WinHello] WinHello.Form1 but

1個の参照
method-id NEW.
procedure division.
  invoke self::InitializeComponent
  goback.
end method.

1個の参照
method-id button1_Click final private.
procedure division using by value sender as object e as type System.EventArgs.
  move "Hello World!" to self::label1::Text.
end method.

end class.

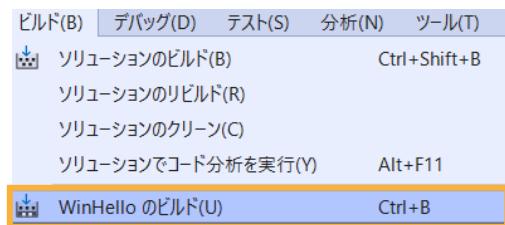
```

11) COBOL アプリケーションをビルドします。

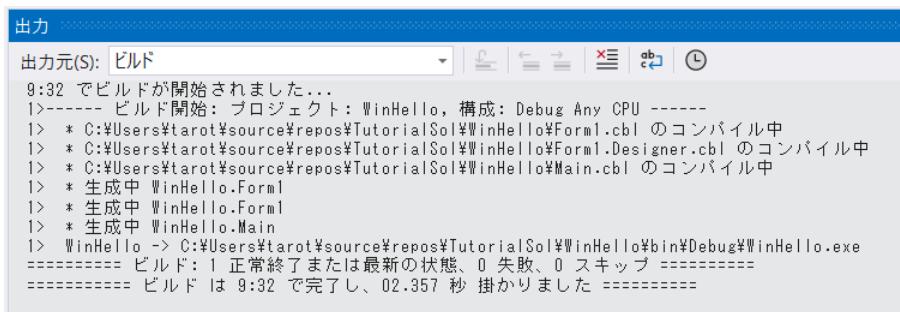
メニューより、[ファイル(F)] > [Form1.cbl の保存] を選択し、変更を保存します。



メニューより、[ビルド(B)] > [WinHello のビルド(U)] を選択します。



出力ウィンドウにビルド結果が表示されますので、ビルドが正常終了したことを確認します。



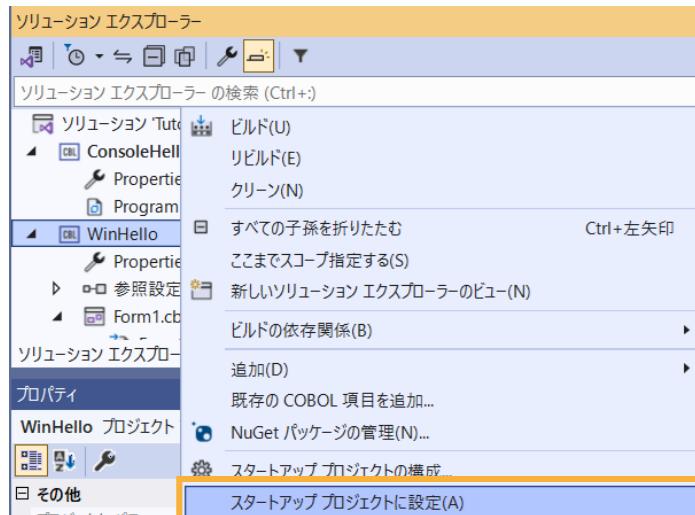
```

出力
出力元(S): ビルド
9:32 でビルドが開始されました...
1>----- ビルド開始: プロジェクト: WinHello, 構成: Debug Any CPU -----
1> * C:\Users\tarot\source\repos\TutorialSol\WinHello\Form1.cbl のコンパイル中
1> * C:\Users\tarot\source\repos\TutorialSol\WinHello\Form1.Designer.cbl のコンパイル中
1> * C:\Users\tarot\source\repos\TutorialSol\WinHello\Main.cbl のコンパイル中
1> * 生成中 WinHello.Form1
1> * 生成中 WinHello.Main
1> WinHello -> C:\Users\tarot\source\repos\TutorialSol\WinHello\bin\Debug\WinHello.exe
===== ビルド: 1 正常終了または最新の状態、0 失敗、0 スキップ =====
===== ビルドは 9:32 で完了し、02.357 秒 掛かりました =====

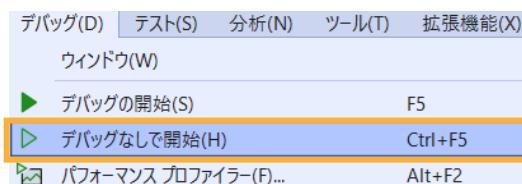
```

12) COBOL アプリケーションを実行します。

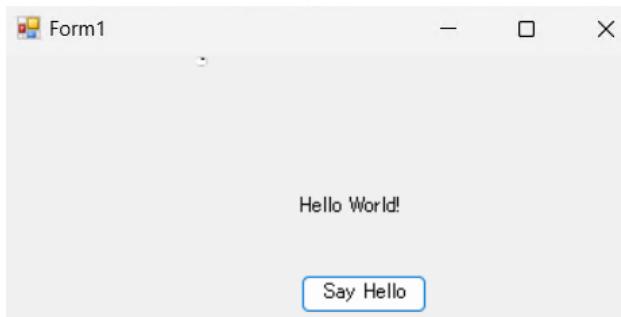
ソリューションエクスプローラーにて WinHello プロジェクトを選択した状態で、マウスの右クリックにてコンテキストメニューを表示し、[スタートアッププロジェクトに設定(A)] を選択します。



メニューより、[デバッグ(D)] > [デバッグなしで開始(H)] を選択すると、Form1 ウィンドウが開きます。



Form1 ウィンドウの [Say Hello] ボタンをクリックすると “Hello World!” が表示されます。



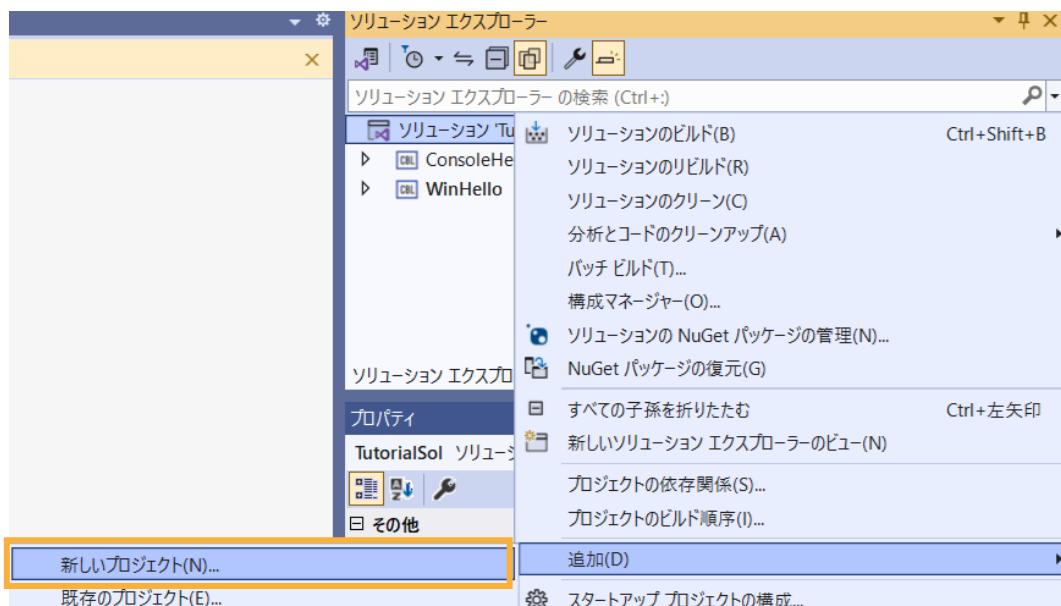
アプリケーションの終了は、[X] ボタンをクリックします。

3.3 ファイルの入出力

続いて、エクセルやメモ帳で作成した CSV ファイルを読み込んで、固定長順編成ファイルを作成する COBOL アプリケーションを作成します。

- 1) 作成したソリューションへプロジェクトを追加します。

さきほどまで使用していたソリューション中のソリューションエクスプローラーにて、ソリューションを選択した状態で、マウスの右クリックにてコンテキストメニューを表示し、[追加(D)] > [新しいプロジェクト(N)...] を選択します。

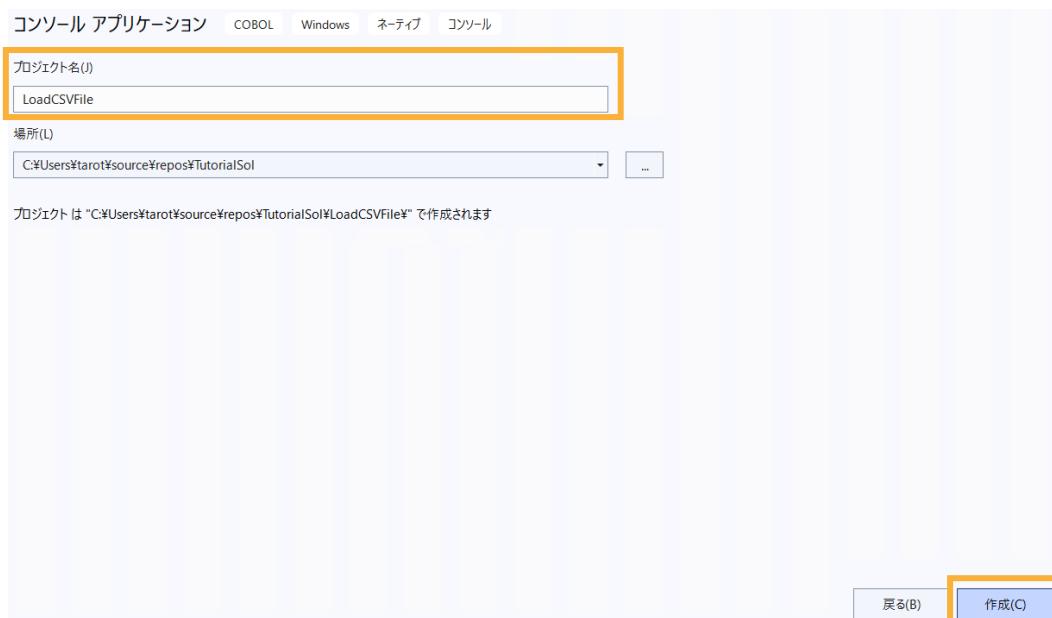


- 2) 使用するテンプレートを選択します。

フィルター画面が表示されるので、言語に “COBOL” 、プラットフォームに “Windows” 、プロジェクト タイプに “ネーティブ” を選択し、一覧から「コンソールアプリケーション」を選んで、[次へ(N)] ボタンをクリックします。



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



3) コードエディターで COBOL ソースコードを入力します。

LoadCSVFile プロジェクトの作成が成功すると、COBOL 専用のコードエディターが起動します。エディター画面にコンソールアプリケーションのひな形が表示されるので、環境部 (environment division)、データ部 (data division)、手続き部(procedure division) を書き換えます。

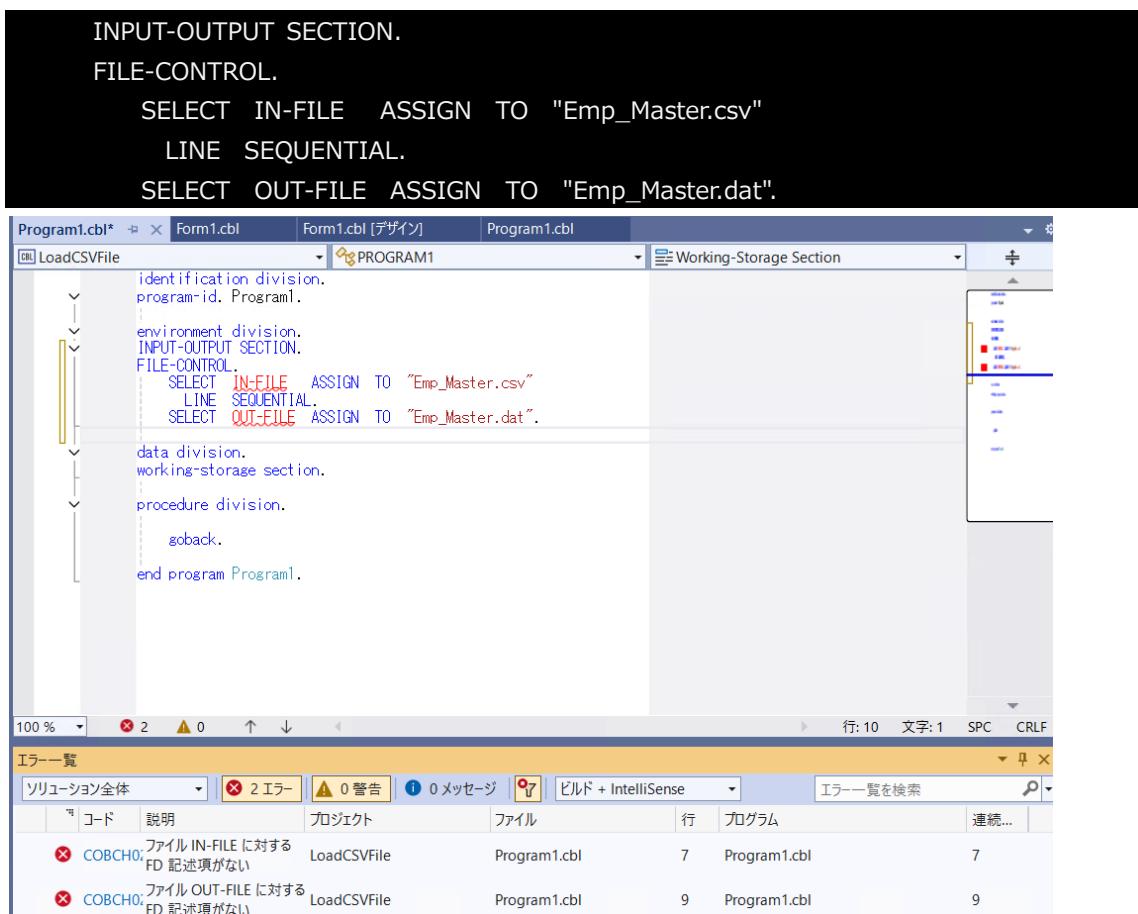
まず、環境部の構成節 (configuration section) を削除し、以下の入出力節 (input-output section) を追加し

ます。まだ、データ部のファイル定義が未入力なので IN-FILE と OUT-FILE がエラーとなります、ここでは無視して構いません。

```

INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT IN-FILE ASSIGN TO "Emp_Master.csv"
    LINE SEQUENTIAL.
  SELECT OUT-FILE ASSIGN TO "Emp_Master.dat".

```

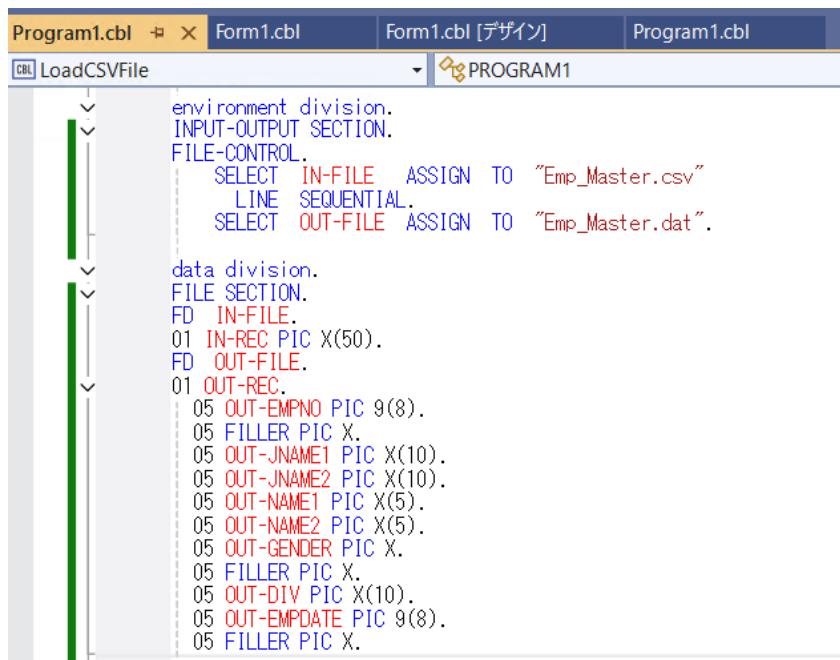


次に、データ部の作業場所節 (working-storage section) を削除し、以下のファイル節 (file section) を追加します。なお、データ部のファイル定義を入力したので、環境部のエラーは無くなります。

```

FILE SECTION.
FD IN-FILE.
  01 IN-REC          PIC X(50).
FD OUT-FILE.
  01 OUT-REC.
    05 OUT-EMPNO      PIC 9(8).
    05 FILLER         PIC X.
    05 OUT-JNAME1     PIC X(10).
    05 OUT-JNAME2     PIC X(10).
    05 OUT-NAME1      PIC X(5).
    05 OUT-NAME2      PIC X(5).
    05 OUT-GENDER     PIC X.
    05 FILLER         PIC X.
    05 OUT-DIV        PIC X(10).
    05 OUT-EMPDAT     PIC 9(8).
    05 FILLER         PIC X.

```



```

Program1.cbl Form1.cbl Form1.cbl [デザイン] Program1.cbl
[COBOL] LoadCSVFile PROGRAM1

environment division.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT IN-FILE ASSIGN TO "Emp_Master.csv"
  LINE SEQUENTIAL.
  SELECT OUT-FILE ASSIGN TO "Emp_Master.dat".

data division.
FILE SECTION.
FD IN-FILE.
01 IN-REC PIC X(50).
FD OUT-FILE.
01 OUT-REC.
  05 OUT-EMPNO PIC 9(8).
  05 FILLER PIC X.
  05 OUT-JNAME1 PIC X(10).
  05 OUT-JNAME2 PIC X(10).
  05 OUT-NAME1 PIC X(5).
  05 OUT-NAME2 PIC X(5).
  05 OUT-GENDER PIC X.
  05 FILLER PIC X.
  05 OUT-DIV PIC X(10).
  05 OUT-EMPCODE PIC 9(8).
  05 FILLER PIC X.

```

最後に、手続き部の goback 文を削除し、以下の 手続き文を追加します。

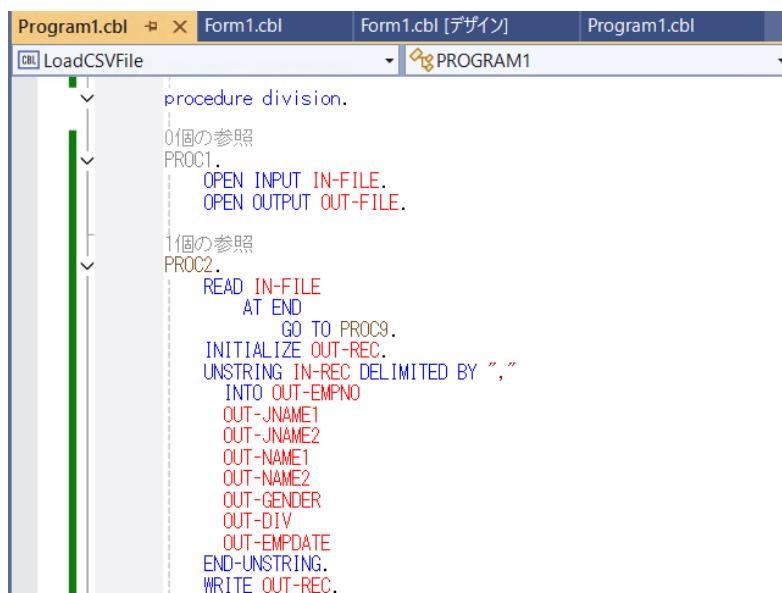
```

PROC1.
  OPEN INPUT IN-FILE.
  OPEN OUTPUT OUT-FILE.

PROC2.
  READ IN-FILE AT END GO TO PROC9.
  INITIALIZE OUT-REC.
  UNSTRING IN-REC DELIMITED BY ","
    INTO OUT-EMPNO
      OUT-JNAME1
      OUT-JNAME2
      OUT-NAME1
      OUT-NAME2
      OUT-GENDER
      OUT-DIV
      OUT-EMPCODE
  END-UNSTRING.
  WRITE OUT-REC.
  GO TO PROC2.

PROC9.
  CLOSE IN-FILE OUT-FILE.
  STOP RUN.

```



```

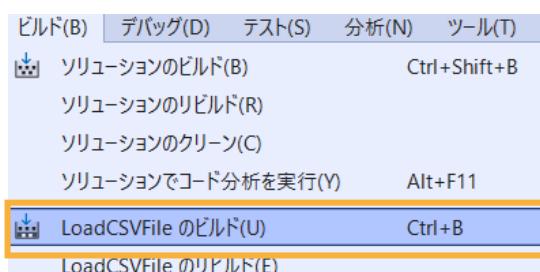
Program1.cbl < X Form1.cbl | Form1.cbl [デザイン] | Program1.cbl
CBL LoadCSVFile PROGAM1
procedure division.
0個の参照
PROC1.
OPEN INPUT IN-FILE.
OPEN OUTPUT OUT-FILE.

1個の参照
PROC2.
READ IN-FILE
AT END
GO TO PROC9.
INITIALIZE OUT-REC.
UNSTRING IN-REC DELIMITED BY ","
INTO OUT-EMPNO
OUT-JNAME1
OUT-JNAME2
OUT-NAME1
OUT-NAME2
OUT-GENDER
OUT-DIV
OUT-EMPODATE
END-UNSTRING.
WRITE OUT-REC.

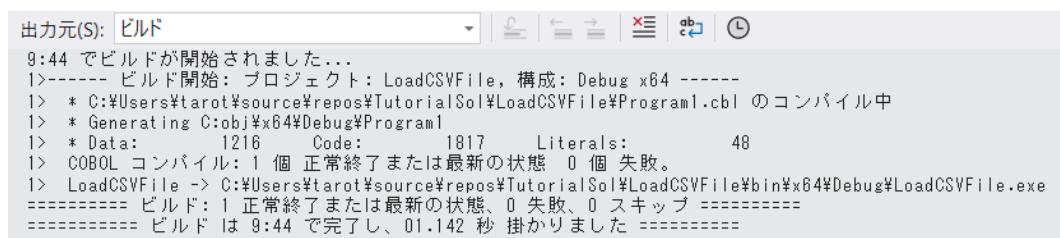
```

- 4) COBOL アプリケーションをビルドします。

メニューより、[ビルド(B)] > [LoadCSVFile のビルド(U)] を選択します。



出力ウィンドウにビルド結果が表示されますので、ビルドが正常終了したことを確認します。



```

出力元(S): ビルド
9:44 でビルドが開始されました...
>----- ビルド開始: プロジェクト: LoadCSVFile, 構成: Debug x64 -----
1> * C:\Users\tarot\source\repos\TutorialSol\LoadCSVFile\Program1.cbl のコンパイル中
1> * Generating C:\obj\x64\Debug\Program1
1> * Data: 1216 Code: 1817 Literals: 48
1> COBOL コンパイル: 1 個 正常終了または最新の状態 0 個 失敗。
1> LoadCSVFile -> C:\Users\tarot\source\repos\TutorialSol\LoadCSVFile\bin\x64\Debug\LoadCSVFile.exe
===== ビルド: 1 正常終了または最新の状態、0 失敗、0 スキップ =====
===== ビルドは 9:44 で完了し、01.142 秒 掛かりました =====

```

- 5) CSV ファイルを作成します。

デバッグフォルダ(<3.2 節の 3) 「場所」で指定したフォルダ> ¥TutorialSol¥LoadCSVFile¥bin¥x64¥debug にメモ帳などを利用して以下の Emp_Master.csv ファイルを作成します。

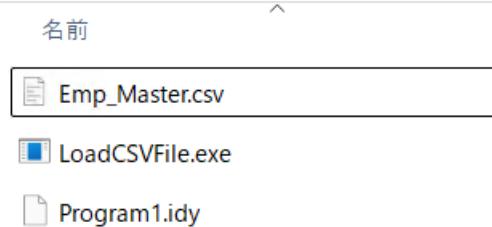
```

11111113,佐藤,隆,サト,タクシ,M,営業部,19980401,0
22222226,鈴木,尚之,スズキ,ナカミ,M,技術部,19981015,0
33333339,田中,直美,タナカ,ナミ,F,総務部,19990401,0
44444442,山田,洋一,ヤマダ,ヨウイチ,M,営業部,20000701,0
55555555,伊藤,弘子,イ藤,ヒロコ,F,技術部,20010401,0
66666668,木村,貴弘,キムラ,タカヒコ,M,営業部,20021220,0
77777771,中村,慎司,ナカムラ,シンジ,M,技術部,20030401,0
88888884,橋本,悦子,ハシモト,エツコ,F,総務部,20040805,0
99999997,三井,薰,ミヅイ,カオル,F,営業部,20050401,0

```

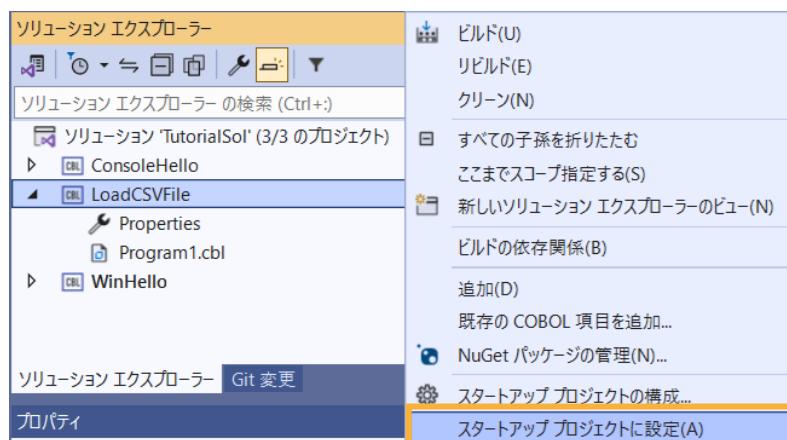
(注意)

上記ファイルは Shift_JIS 形式で保存してください。メモ帳を使用する場合、Windows 製品のバージョンによって保存する文字コード形式が異なりますので、ANSI を選択してください。



- 6) COBOL アプリケーションをデバッグ実行します。

ソリューションエクスプローラーにて LoadCSVFile プロジェクトを選択した状態で、マウスの右クリックにてコンテキストメニューを表示し、[スタートアッププロジェクトに設定(A)] を選択します。



続いて、メニューより、[デバッグ(D)] > [ステップイン(I)] を選択して、デバッガーによるステップ実行を開始します。デバッガーは手続き部の最初の COBOL 文である open 文で実行を中断します。



```

procedure division.

0個の参照
PROC1.
  OPEN INPUT IN-FILE.
  OPEN OUTPUT OUT-FILE.

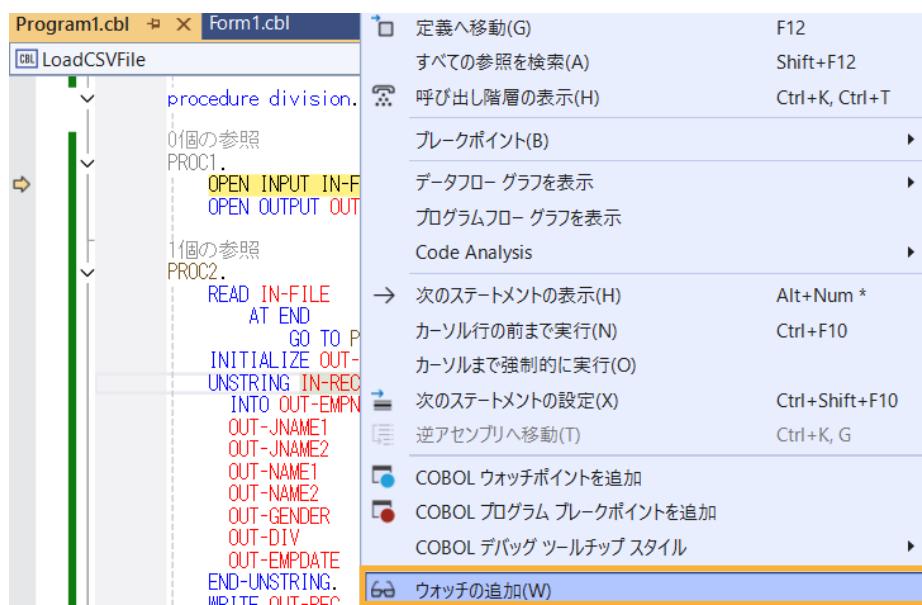
1個の参照
PROC2.
  READ IN-FILE
  AT END
    GO TO PROC9.
  INITIALIZE OUT-REC.
  UNSTRING IN-REC DELIMITED BY ","
    INTO OUT-FMPNO

```

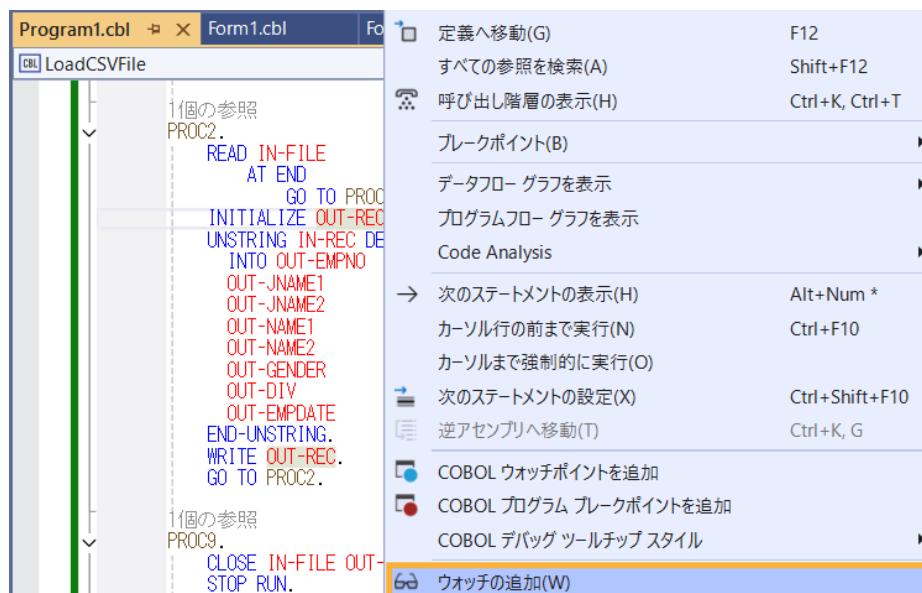
(補足)

ステップインには、F11 キーがホットキーとして割り当てられているため、F11 キーを押すことでステップイン指示が可能です。後述する作業では F11 キーを使用してステップイン指示を実施します。

入力ファイルから読み込んだレコードの内容を確認するため、unstring 文の in-rec 上でマウスの右クリックにてコンテキストメニューを表示し、[ウォッチの追加(W)] を選択します。



同様に出力ファイルに書き出すレコードの内容を確認するため、initialize 文の out-rec 上でもマウスの右クリックにてコンテキストメニューを表示し、[ウォッチの追加(W)] を選択します。



F11 キーを 3 回押すと、デバッガーは read 文実行後、処理を中断します。ウォッチ式の in-rec の値には CSV ファイルから読み込んだ最初のレコードが表示されます。

Program1.cbl Form1.cbl Form1.cbl [デザイン] Program1.cbl

LoadCSVFile PROGRAM1

```

05 OUT-GENDER PIC X.
05 FILLER PIC X.
05 OUT-DIV PIC X(10).
05 OUT-EMPCODE PIC 9(8).
05 FILLER PIC X.

procedure division.

0個の参照
PROC1.
    OPEN INPUT IN-FILE.
    OPEN OUTPUT OUT-FILE.

1個の参照
PROC2.
    READ IN-FILE
    AT END
        GO TO PROC9.
    INITIALIZE OUT-REC.
    UNSTRING IN-REC DELIMITED BY ","
        INTO OUT-EMPNO
        OUT-JNAME1
        OUT-JNAME2
        OUT-TNAME1

```

100 % 問題は見つかりませんでした

ウォッチ 1

名前	値	種類
IN-REC	11111113,佐藤,隆,サトウ,タケシ,M,営業部,19980401...	PIC X(50)
OUT-REC	{長さ=60} : "	GROUP

項目をウォッチに追加する

自動 ローカル ウォッチ 1

さらに F11 キーを 2 回押すと、デバッガーは unstring 文を実行後、処理を中断します。ウォッチ式の out-rec の値には出力ファイルへ書き出す最初のレコードが表示されます。

Program1.cbl Form1.cbl Form1.cbl [デザイン] Program1.cbl

LoadCSVFile PROGRAM1

```

INITIALIZE OUT-REC.
UNSTRING IN-REC DELIMITED BY ","
  INTO OUT-EMPNO
  OUT-JNAME1
  OUT-JNAME2
  OUT-NAME1
  OUT-NAME2
  OUT-GENDER
  OUT-DIV
  OUT-EMPDATE
END-UNSTRING.
WRITE OUT-REC.
GO TO PROC2.

1個の参照
PROC9.
CLOSE IN-FILE OUT-FILE.
STOP RUN.

end program Program1.

```

100 % 問題は見つかりませんでした

ウォッチ 1

名前	値	種類
IN-REC	11111113,佐藤,隆,サウタカシ,M,営業部,19980401...	PIC X(50)
OUT-REC	{長さ=60} : "1111113 佐藤 隆 サウタカシ,M,営業部,19980401..."	GROUP

さらに F11 キーを 4 回押すと、デバッガーは initialize 文を実行後、処理を中断します。ウォッチ式の in-rec の値には CSV ファイルから読み込んだ 2 番目のレコードが表示され、out-rec の値は initialize 文で初期化されています。

Program1.cbl Form1.cbl Form1.cbl [デザイン] Program1.cbl

LoadCSVFile PROGRAM1

```

procedure division.
0J FILLER PIC X(50).

OPEN INPUT IN-FILE.
OPEN OUTPUT OUT-FILE.

1個の参照
PROC1.
READ IN-FILE
  AT END
  GO TO PROC9.

INITIALIZE OUT-REC.
UNSTRING IN-REC DELIMITED BY ","
  INTO OUT-EMPNO
  OUT-JNAME1
  OUT-JNAME2
  OUT-NAME1
  OUT-NAME2
  OUT-GENDER
  OUT-DIV
  OUT-EMPDATE
END-UNSTRING.

```

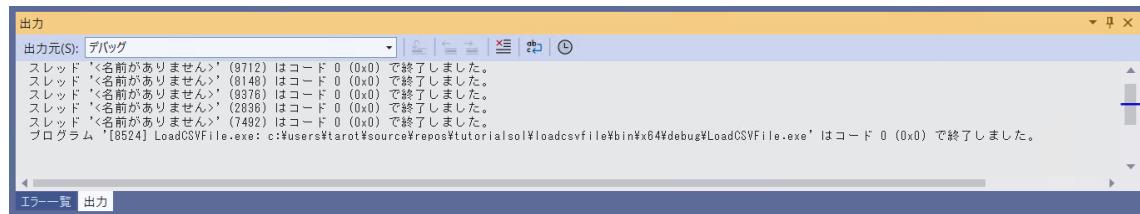
100 % 問題は見つかりませんでした

ウォッチ 1

名前	値	種類
IN-REC	22222226,鈴木,尚之,スミヤオノ, M,技術部,1998...	PIC X(50)
OUT-REC	{長さ=60} : "00000000"	GROUP

自動 ローカル ウォッチ 1

メニューより、[デバッグ(D)] > [続行(C)] を選択するか、CSV ファイルからすべてのレコードを読み込むまで F11 キーを押すと、デバッガーは終了します。



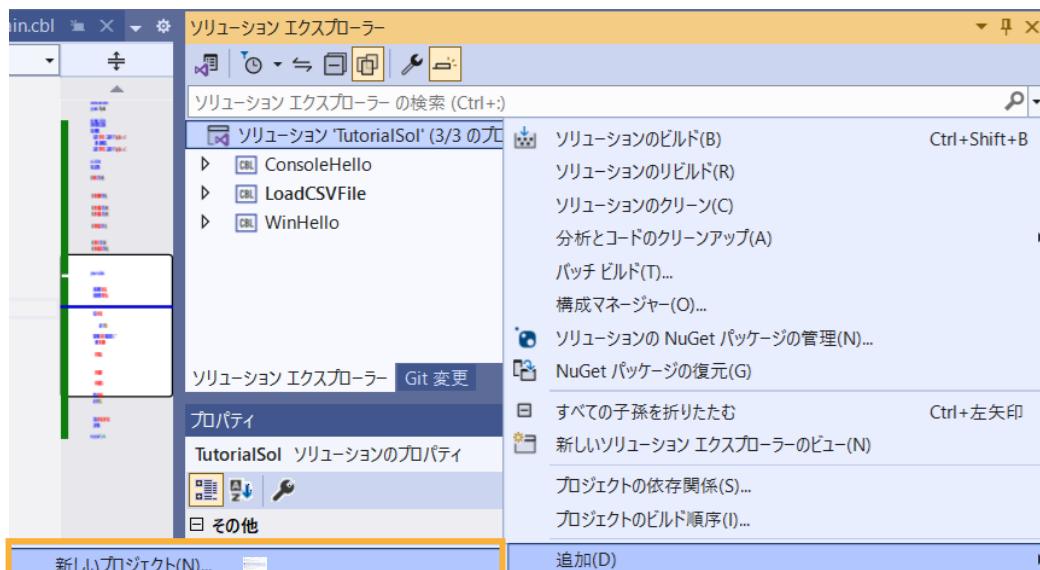
さきほど Emp_Master.csv を作成したフォルダ配下に Emp_Master.dat ファイルが作成されます。テキストエディタなどでファイルを開き、社員 9 名分のデータが表示されることを確認します。エディター上で 60 行で折り返し設定を行うと、以下のように表示されます。

0	1	2	3	4	5
11111113	佐藤 隆	サトウ タカシ	M	営業部	19980401
22222226	鈴木 尚之	スズキ ナオユキ	M	技術部	19981015
33333339	田中 直美	タナカ ナオミ	F	総務部	19990401
44444442	山田 洋一	ヤマタ ヨウイチ	M	営業部	20000701
55555555	伊藤 弘子	イトウ ヒロコ	F	技術部	20010401
66666668	木村 貴弘	キムラ タカヒロ	M	営業部	20021220
77777771	中村 慎司	ナカムラ シンジ	M	技術部	20030401
88888884	橋本 悅子	ハシモト エイコ	F	総務部	20040805
99999997	三井 薫	ミツイ カオル	F	営業部	20050401

さきほどの固定長順編成ファイルを読み込んでレポートファイルを作成するパッチアプリケーションを作成します。

7) ソリューションに新規プロジェクトを追加します。

ソリューションエクスプローラーにて、TutorialSol ソリューションを選択した状態で、マウスの右クリックにてコンテキストメニューを表示し、[追加(D)] > [新しいプロジェクト(N)...] を選択します。



8) 使用するテンプレートを選択します。

フィルター画面が表示されるので、言語に “COBOL”、プラットフォームに “Windows”、プロジェクト タイプに “ネーティブ” を選択し、一覧から「コンソールアプリケーション」を選んで、[次へ(N)] ボタンをクリックします。

COBOL Windows ネーティブ

Windows アプリケーション
Windows アプリケーションを作成するためのネイティブ プロジェクトです。

COBOL Windows ネーティブ

コンソール アプリケーション
ネイティブ コマンドライン アプリケーションを作成するためのプロジェクトです。

COBOL Windows ネーティブ コンソール

空のプロジェクト
ローカル アプリケーションを作成するための空のプロジェクトです。

COBOL Windows ネーティブ

リンク ライブラリ
他のアプリケーションで使用するクラスを作成するためのネイティブ プロジェクトです。

COBOL Windows ネーティブ ライブラリ

ユニット テスト ライブラリ
MFUnit テスト ライブラリを作成するためのネイティブ プロジェクトです。

COBOL Windows ネーティブ テスト ライブラリ

Micro Focus INT/GNT
Micro Focus INT または GNT コードを作成するためのプロジェクトです。

次へ(N)

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

コンソール アプリケーション COBOL Windows ネーティブ コンソール

プロジェクト名(I)
BATCHRPT

場所(L)
C:\Users\tarot\source\repos\TutorialSol

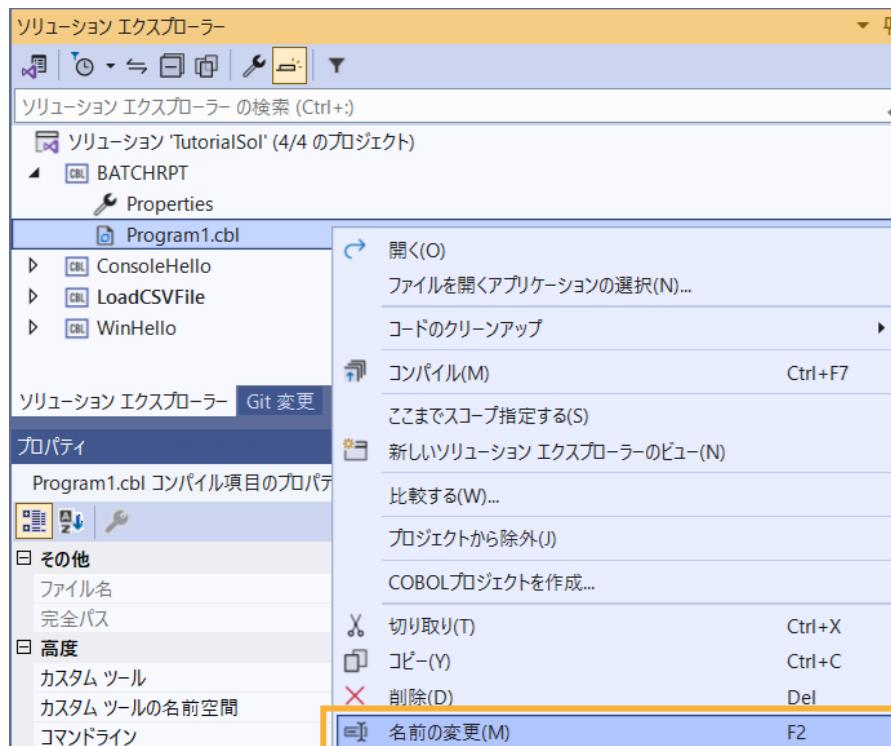
プロジェクトは "C:\Users\tarot\source\repos\TutorialSol\BATCHRPT" で作成されます

戻る(B) 作成(C)

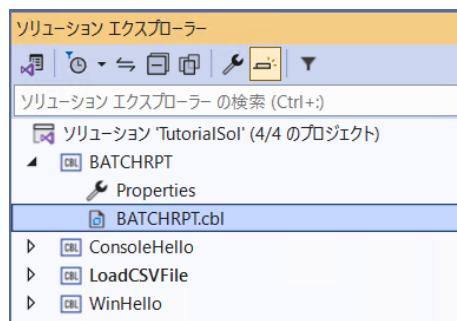
10) コードエディターで COBOL ソースコードを入力します。

BATCHRPT プロジェクトの作成が成功すると、COBOL 専用のコードエディターが起動します。エディター画面にコンソ-

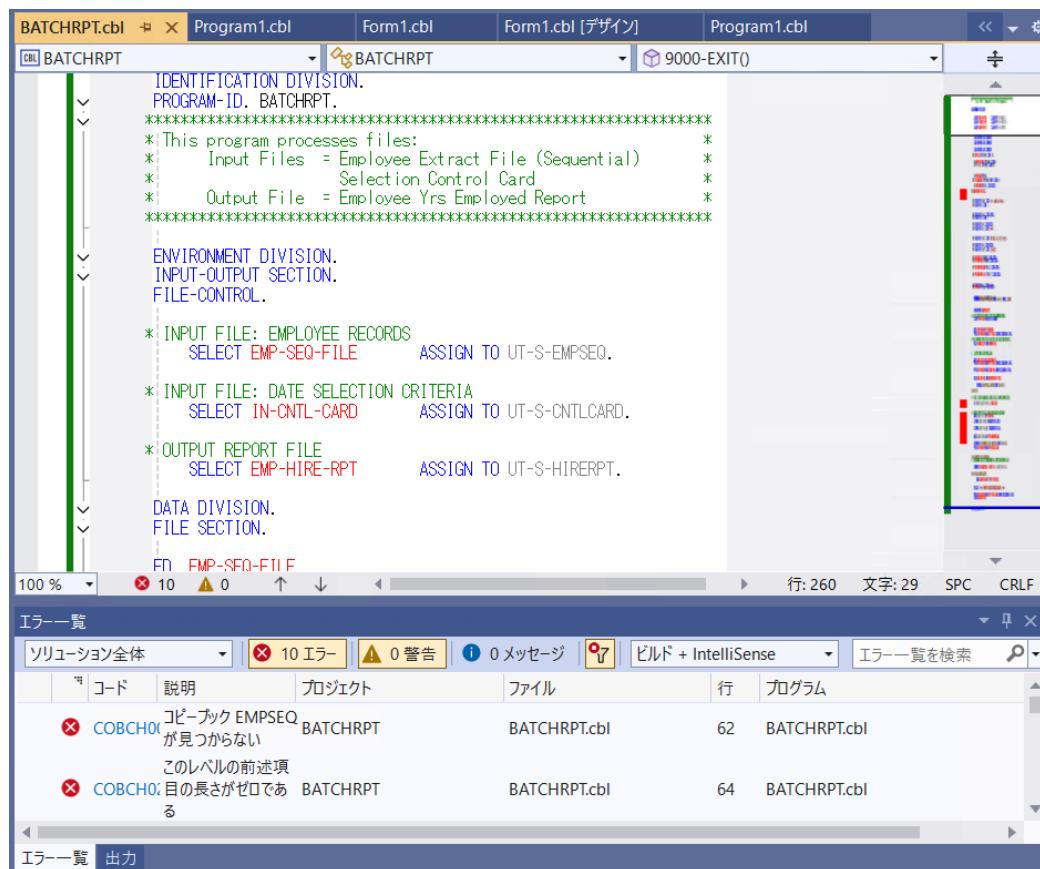
ルアーカイションのひな形が表示されるので、ソリューションエクスプローラーで BATCHRPT プロジェクト配下のソースプログラム「Program1.cbl」を選択した上で、マウスの右クリックにてコンテキストメニューを表示し、[名前の変更(M)] を選択します。その後、プログラム名を “BATCHRPT.cbl” に書き換えます。



変更後のソリューションエクスプローラー



サンプルプログラム BARCHRPT.cbl の内容でコードを上書き保存します。



```

IDENTIFICATION DIVISION.
PROGRAM-ID. BATCHRPT.
***** This program processes files: *****
*      Input Files = Employee Extract File (Sequential) *
*      Selection Control Card *
*      Output File = Employee Yrs Employed Report *
***** ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

* INPUT FILE: EMPLOYEE RECORDS
  SELECT EMP-SEQ-FILE    ASSIGN TO UT-S-EMPSEQ.

* INPUT FILE: DATE SELECTION CRITERIA
  SELECT IN-CNTL-CARD   ASSIGN TO UT-S-CNTLCARD.

* OUTPUT REPORT FILE
  SELECT EMP-HIRE-RPT   ASSIGN TO UT-S-HIRERPT.

DATA DIVISION.
FILE SECTION.

FN  FMP-SFO-FILE

```

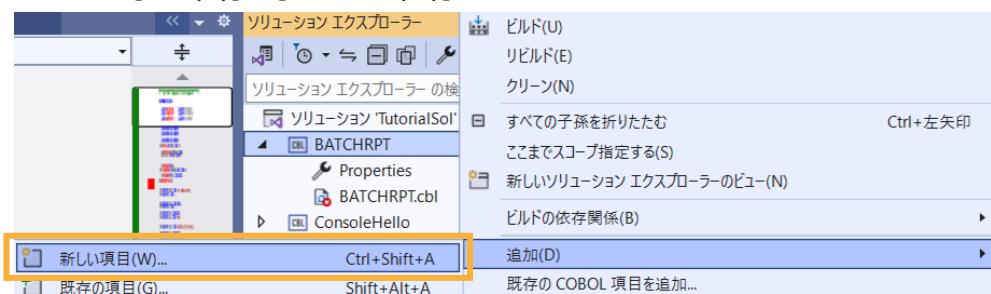
行: 260 文字: 29 SPC CRLF

エラー一覧

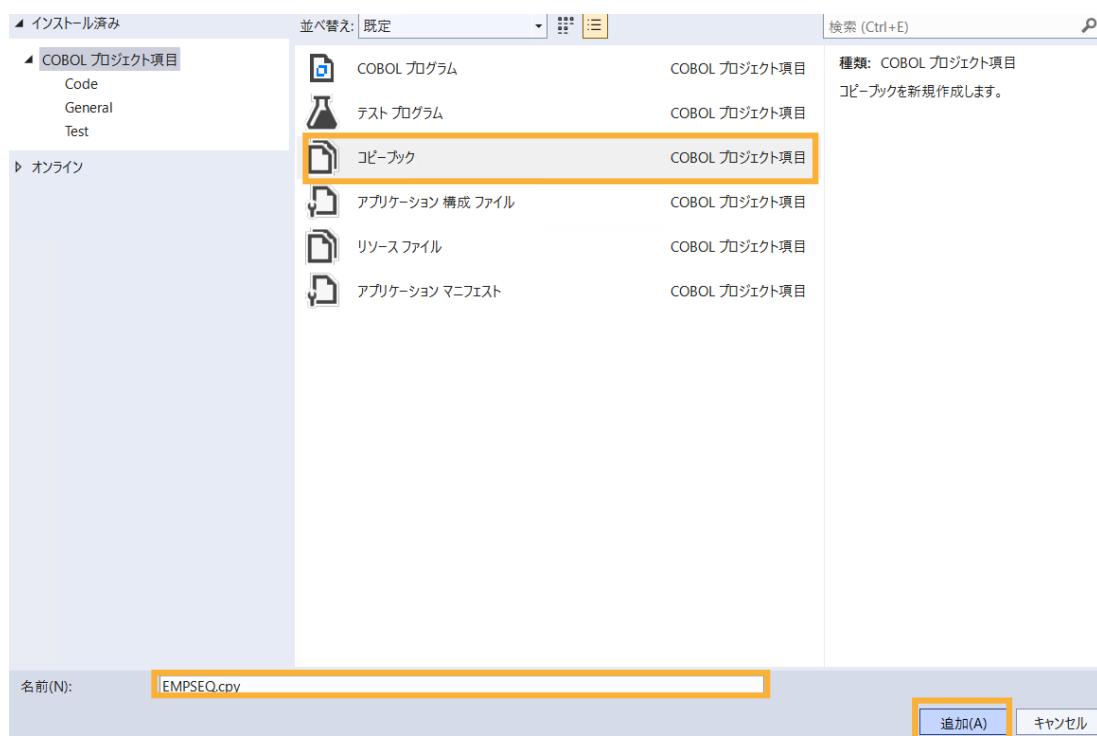
ソリューション全体	コード	説明	プロジェクト	ファイル	行	プログラム
	✖ COBCHOK	コピー・ブック EMPSEQ が見つからない	BATCHRPT	BATCHRPT.cbl	62	BATCHRPT.cbl
	✖ COBCHOK:	このレベルの前述項				
		目の長さがゼロである	BATCHRPT	BATCHRPT.cbl	64	BATCHRPT.cbl

参照しているコピー・ブックが存在しないため、エラーが報告されますが、ここでは無視して構いません。
続いて、参照されるコピー・ブックを作成します。

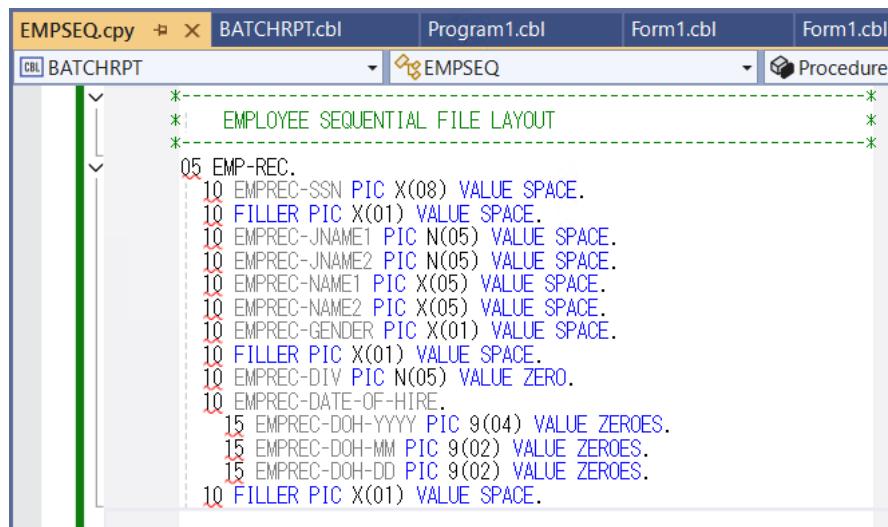
TutorialSol ソリューション配下の BARCHRPT プロジェクト名を選択した上で、マウスの右クリックにてコンテキストメニューを表示し、[追加(D)] > [新しい項目(W)] を選択します。



「コピー・ブック」を選択し、名前に “EMPSEQ.cpy” を入力し、[追加(A)] ボタンをクリックします。



作成された EMPSEQ.cpy の中身を、サンプルプログラム EMPSEQ.cbl で上書き保存します。

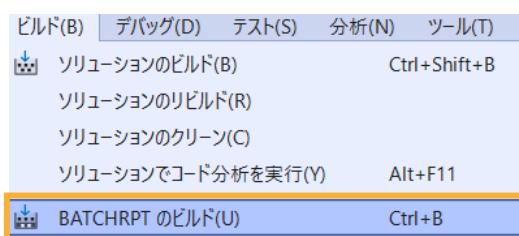


```

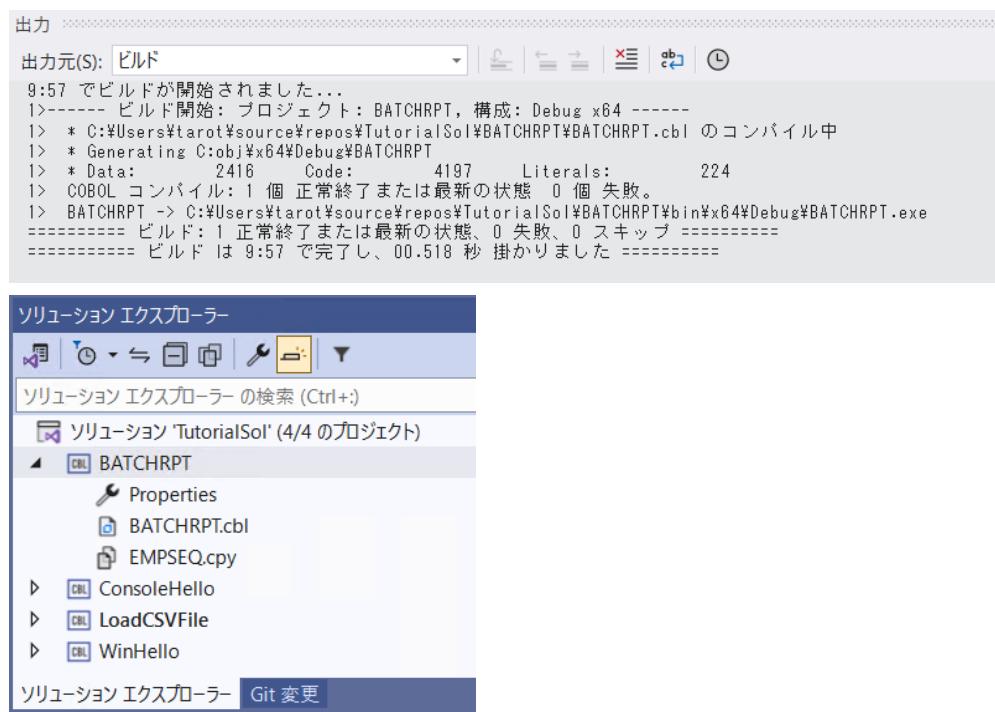
*-----*
*|  EMPLOYEE SEQUENTIAL FILE LAYOUT
*-----*
*| 05 EMP-REC.
*| 10 EMPREC-SSN PIC X(08) VALUE SPACE.
*| 10 FILLER PIC X(01) VALUE SPACE.
*| 10 EMPREC-JNAME1 PIC N(05) VALUE SPACE.
*| 10 EMPREC-JNAME2 PIC N(05) VALUE SPACE.
*| 10 EMPREC-NAME1 PIC X(05) VALUE SPACE.
*| 10 EMPREC-NAME2 PIC X(05) VALUE SPACE.
*| 10 EMPREC-GENDER PIC X(01) VALUE SPACE.
*| 10 FILLER PIC X(01) VALUE SPACE.
*| 10 EMPREC-DIV PIC N(05) VALUE ZERO.
*| 10 EMPREC-DATE-OF-HIRE.
*| 15 EMPREC-DOH-YYYY PIC 9(04) VALUE ZEROES.
*| 15 EMPREC-DOH-MM PIC 9(02) VALUE ZEROES.
*| 15 EMPREC-DOH-DD PIC 9(02) VALUE ZEROES.
*| 10 FILLER PIC X(01) VALUE SPACE.

```

メニューより、[ビルド(B)] > [BATCHRPT のビルド(U)] を選択します。

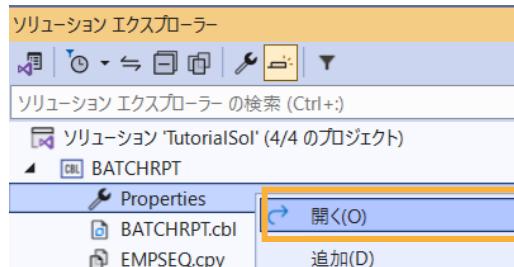


ビルドが正常終了することを確認します。

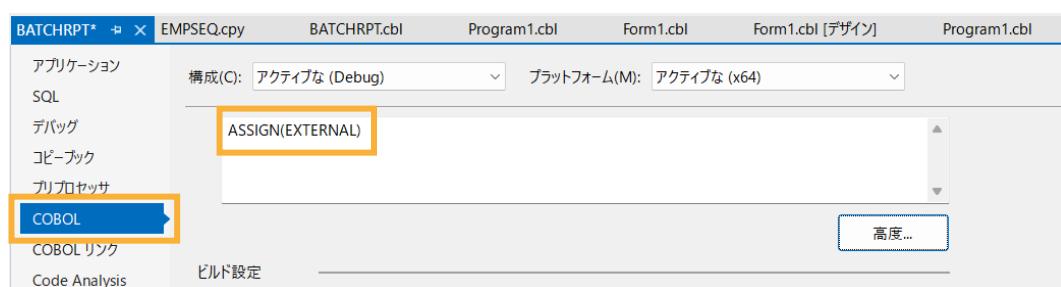


11) COBOL コンパイル指令を追加します。

ファイル名の割り当てを EXTERNAL (外部割り当て) に変更するため、ソリューションエクスプローラーにて BATCHRPT プロジェクト配下の Properties を選択し、マウスの右クリックにてコンテキストメニューを表示し、[開く(O)] を選択します。

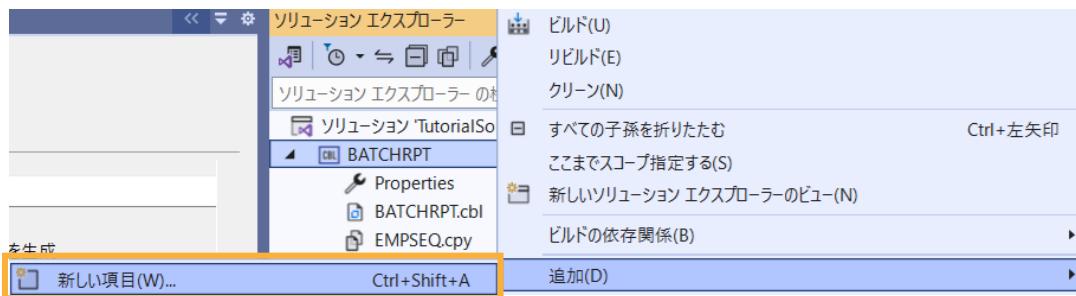


COBOL タブを選択し 追加指令に “ASSIGN(EXTERNAL)” を入力し、変更を保存します。



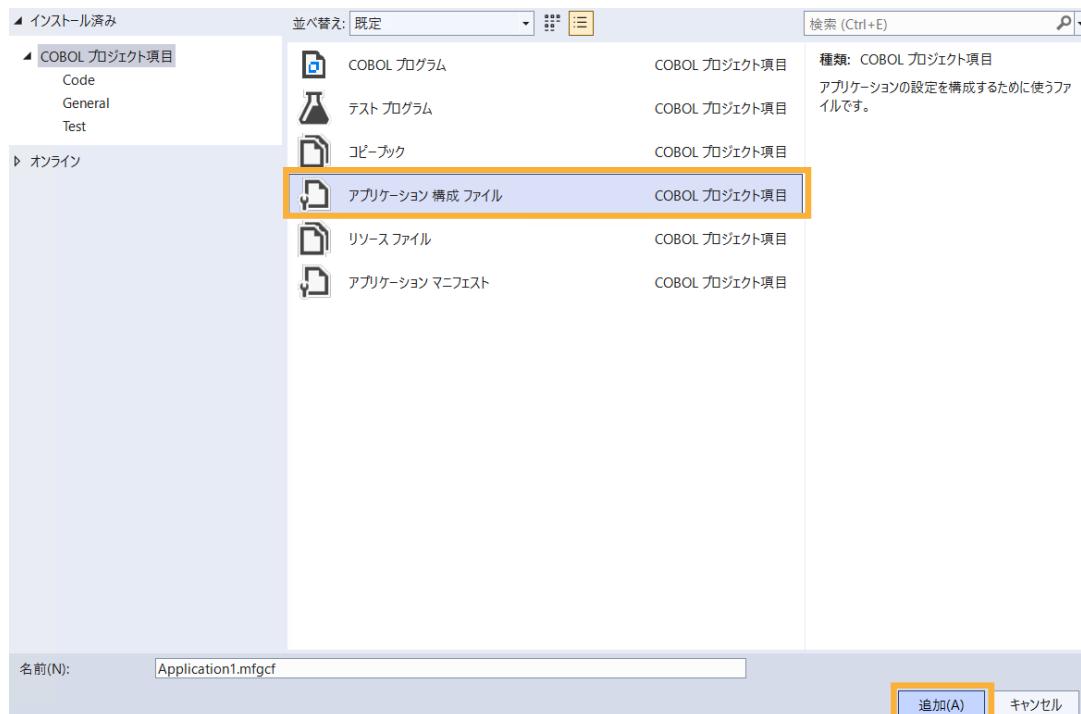
12) アプリケーション構成ファイルを作成します。

TutorialSol ソリューション配下の BARCHRPT プロジェクト名を選択し、マウスの右クリックにてコンテキストメニューを表示し、[追加(D)] > [新しい項目(W)] を選択します。



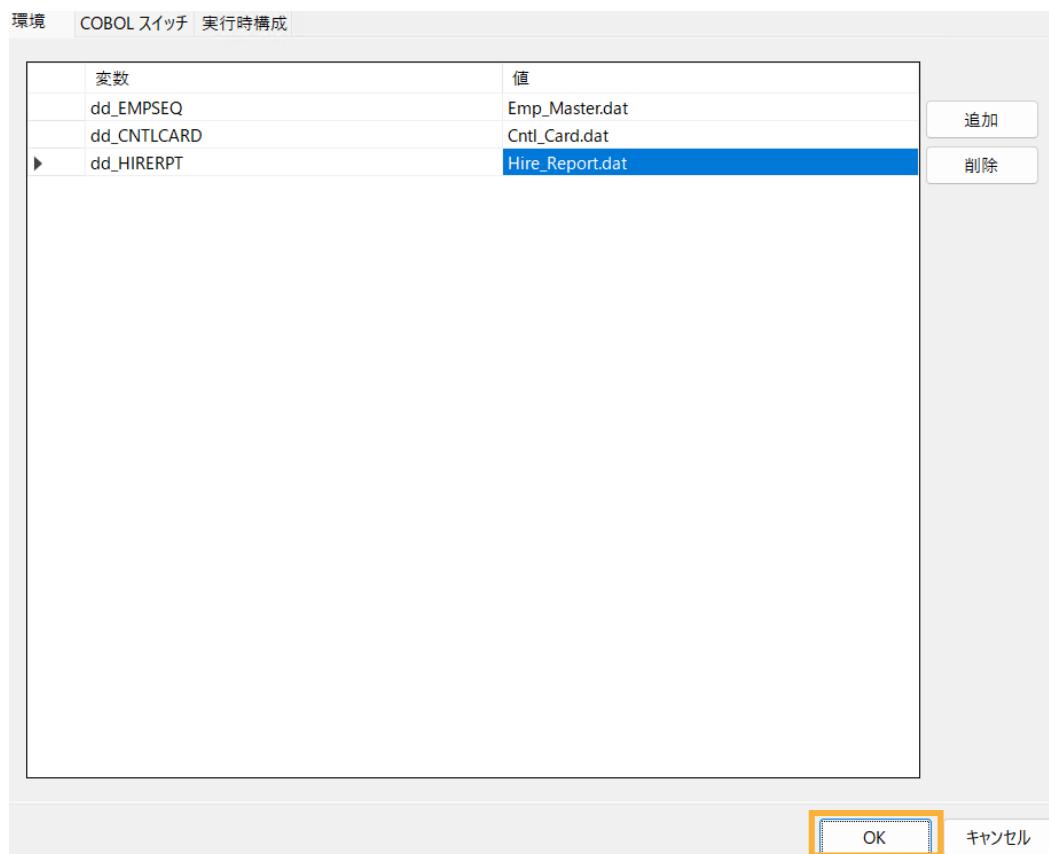
13) 使用するテンプレートを選択します。

「アプリケーション構成ファイル」を選択し、[追加(A)] をクリックします。



生成された Application1.mfgcf ファイルをダブルクリックし、表示されたダイアログで [追加] をクリックし、以下の入力を行います。

変数	値
dd_EMPSEQ	Emp_Master.dat
dd_CNTLCARD	Cntl_Card.dat
dd_HIRERPT	Hire_Report.dat



上記のようになっていることを確認のうえ、[OK] をクリックします。

14) COBOL アプリケーションをビルドします。

メニューより、[ビルド(B)] > [BATCHRPT のリビルド(E)] を選択します。



出力ウィンドウにビルド結果が表示されるので、すべてのビルドが正常終了したことを確認します。

```
出力元(S): ビルド
10:03 で再構築が開始されました...
1>----- すべてのリビルド開始: プロジェクト:BATCHRPT, 構成: Debug x64 -----
1> * C:\Users\tarot\source\repos\TutorialSol\BATCHRPT\BATCHRPT.cbl のコンパイル中
1> * Generating C:\obj\x64\Debug\BATCHRPT
1> * Data: 2400 Code: 4197 Literals: 224
1> COBOL コンパイル: 1 個 正常終了または最新の状態 0 個 失敗。
1> BATCHRPT -> C:\Users\tarot\source\repos\TutorialSol\BATCHRPT\bin\x64\Debug\BATCHRPT.exe
===== すべて再構築: 1 正常終了、0 失敗、0 スキップ =====
===== リビルドは 10:03 で完了し、00.439 秒 掛かりました =====
```

15) 入力ファイルをコピーします。

手順 6)の最後で作成された Emp_Master.dat ファイルをデバッグフォルダ（<3.2 節の 3）「場所」で指定したフォルダ（¥TutorialSol¥BATCHRPT¥bin¥x64¥debug）にコピーします。



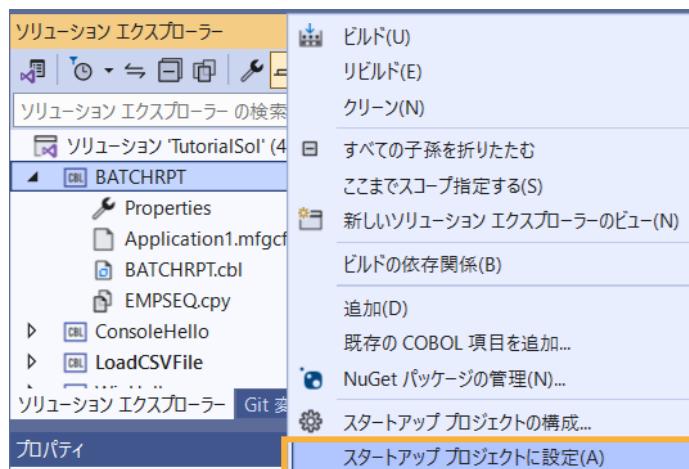
16) 制御ファイルを作成します。

“20000101”が記載された Cntl_Card.dat ファイルをデバッグフォルダ（<3.2 節の 3）「場所」で指定したフォルダ（¥TutorialSol¥BATCHRPT¥bin¥x64¥debug）に作成します。

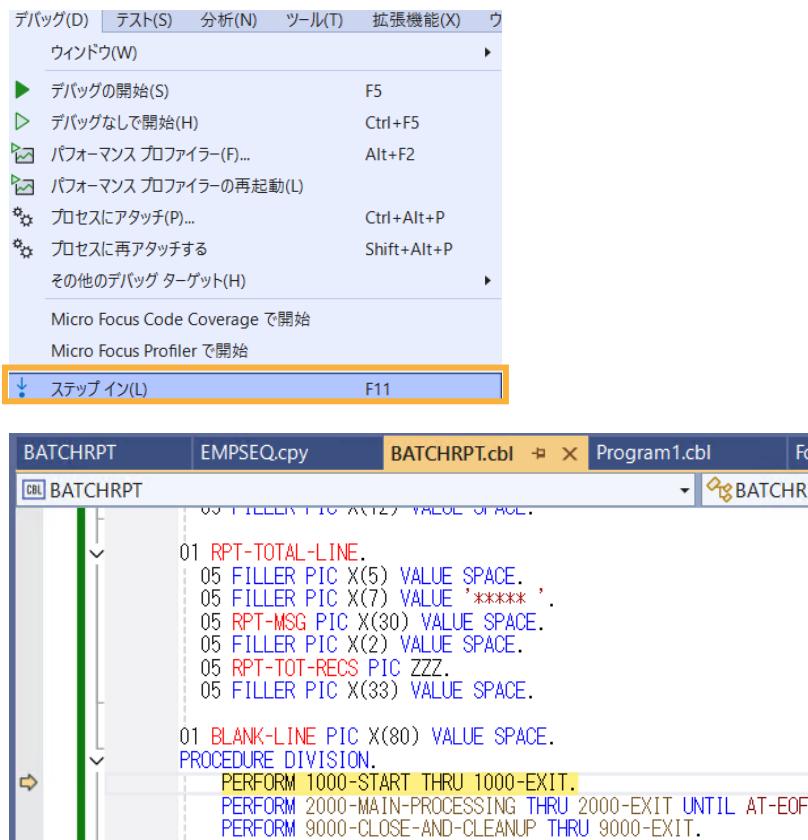


17) COBOL アプリケーションをデバッグ実行します。

ソリューションエクスプローラーにて BATCHRPT プロジェクトを選択した状態で、マウスの右クリックにてコンテキストメニューを表示し、[スタートアッププロジェクトに設定(A)] を選択します。



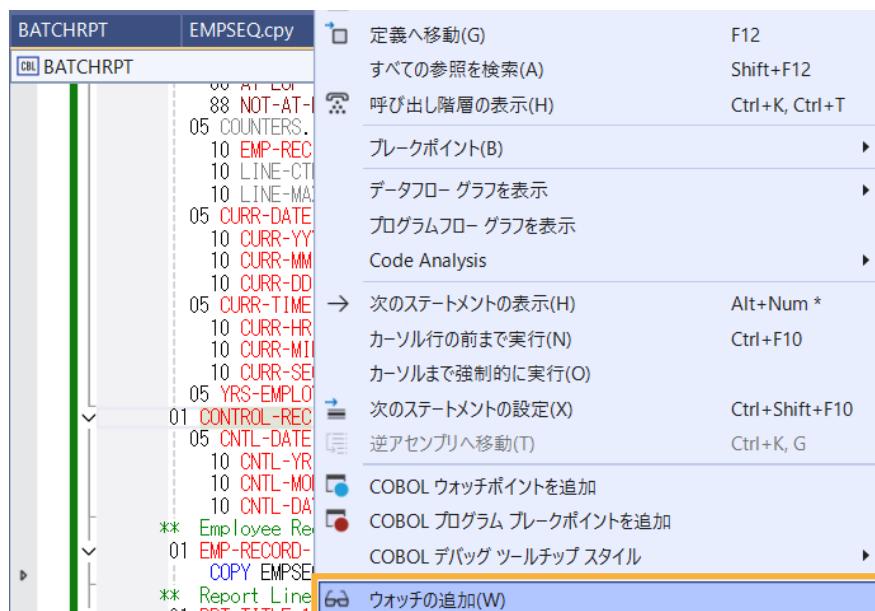
メニューより、[デバッグ(D)] > [ステップイン(I)] を選択するか F11 キーを押すと、コマンドプロンプト画面が開き、デバッガがステップ実行を開始します。デバッガーは手続き部の最初の COBOL 文である PERFORM 文を実行する手前で処理を中断します。



The screenshot shows the Micro Focus COBOL IDE interface. The menu bar at the top includes 'デバグ(D)', 'テスト(S)', '分析(N)', 'ツール(T)', '拡張機能(X)', and 'ウ' (Window). A context menu is open over the COBOL code, specifically over the 'CONTROL-REC' record definition. The menu items include:

- 定義へ移動(G) F12
- すべての参照を検索(A) Shift+F12
- 呼び出し階層の表示(H) Ctrl+K, Ctrl+T
- ブレークポイント(B)
- データフロー グラフを表示
- プログラムフロー グラフを表示
- Code Analysis
- 次のステートメントの表示(H) Alt+Num *
- カーソル行の前まで実行(N) Ctrl+F10
- カーソルまで強制的に実行(O)
- 次のステートメントの設定(X) Ctrl+Shift+F10
- 逆アセンブリへ移動(T) Ctrl+K, G
- COBOL ウオッチポイントを追加
- COBOL プログラム ブレークポイントを追加
- COBOL デバッグ ツールチップスタイル
- ウォッチの追加(W)**

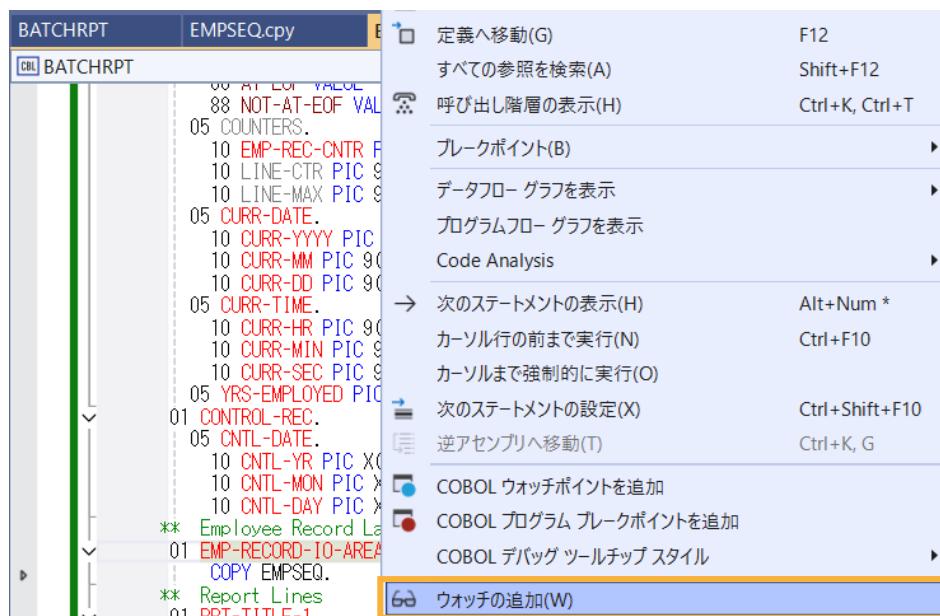
制御ファイルから読み込んだレコードの内容を確認するため、データ部の CONTROL-REC 上でマウスの右クリックにてコンテキストメニューを表示し、[ウォッチの追加(W)] を選択します。



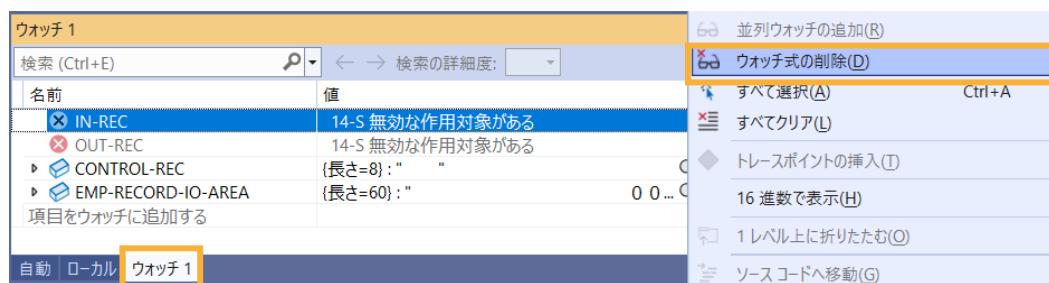
The screenshot shows the Micro Focus COBOL IDE interface. The menu bar at the top includes 'デバグ(D)', 'テスト(S)', '分析(N)', 'ツール(T)', '拡張機能(X)', and 'ウ' (Window). A context menu is open over the COBOL code, specifically over the 'EMP-RECORD' record definition. The menu items include:

- 定義へ移動(G) F12
- すべての参照を検索(A) Shift+F12
- 呼び出し階層の表示(H) Ctrl+K, Ctrl+T
- ブレークポイント(B)
- データフロー グラフを表示
- プログラムフロー グラフを表示
- Code Analysis
- 次のステートメントの表示(H) Alt+Num *
- カーソル行の前まで実行(N) Ctrl+F10
- カーソルまで強制的に実行(O)
- 次のステートメントの設定(X) Ctrl+Shift+F10
- 逆アセンブリへ移動(T) Ctrl+K, G
- COBOL ウオッチポイントを追加
- COBOL プログラム ブレークポイントを追加
- COBOL デバッグ ツールチップスタイル
- ウォッチの追加(W)**

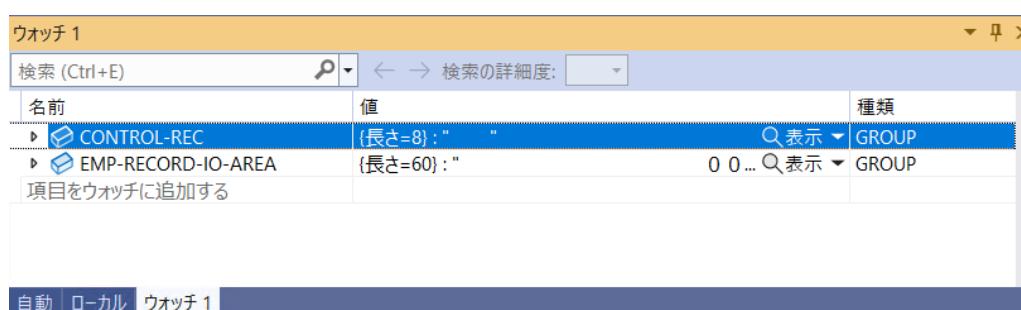
同様に入力ファイルから読み込んだレコードの内容を確認するため、データ部の EMP-RECORD-IO-AREA 上でマウスの右クリックにてコンテキストメニューを表示し、[ウォッチの追加(W)] を選択します。



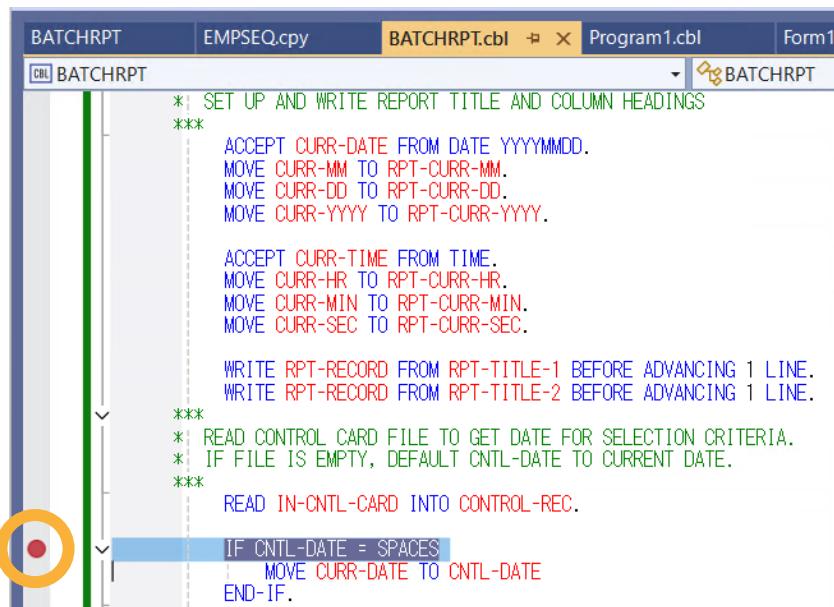
前回のデバッグ作業で追加したウォッチ項目を削除します。画面下部より [ウォッチ 1] タブを選択し、[IN-REC] を選択、マウスの右クリックでコンテキストメニューを開き、[ウォッチ式の削除(D)] を選択します。



同様に、OUT-REC 項目もウォッチ式の削除を行ってください。



手続き部 1000-START 節の READ 文に続く IF 文でエディター画面の左端をクリックし、ブレークポイントを設定します。



```

BATCHRPT EMPSEQ.cpy BATCHRPT.cbl Program1.cbl Form1
CBL BATCHRPT
* SET UP AND WRITE REPORT TITLE AND COLUMN HEADINGS
***  

ACCEPT CURR-DATE FROM DATE YYYYMMDD.  

MOVE CURR-MM TO RPT-CURR-MM.  

MOVE CURR-DD TO RPT-CURR-DD.  

MOVE CURR-YYYY TO RPT-CURR-YYYY.  

ACCEPT CURR-TIME FROM TIME.  

MOVE CURR-HR TO RPT-CURR-HR.  

MOVE CURR-MIN TO RPT-CURR-MIN.  

MOVE CURR-SEC TO RPT-CURR-SEC.  

WRITE RPT-RECORD FROM RPT-TITLE-1 BEFORE ADVANCING 1 LINE.  

WRITE RPT-RECORD FROM RPT-TITLE-2 BEFORE ADVANCING 1 LINE.  

***  

* READ CONTROL CARD FILE TO GET DATE FOR SELECTION CRITERIA.  

* IF FILE IS EMPTY, DEFAULT CTLN-DATE TO CURRENT DATE.  

***  

READ IN-CTLN-CARD INTO CONTROL-REC.  

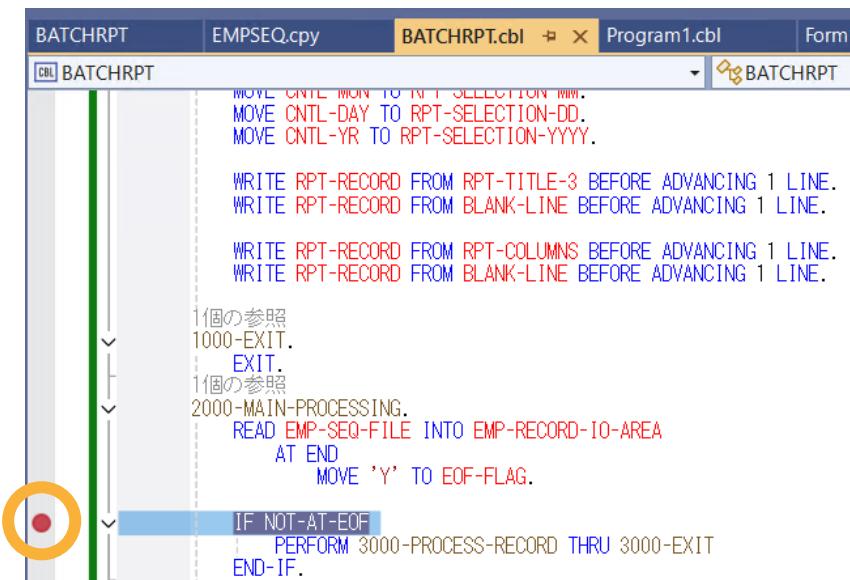
IF CTLN-DATE = SPACES  

    MOVE CURR-DATE TO CTLN-DATE  

END-IF.

```

同様に手続き部 2000-MAIN-PROCESSING 段落の READ 文に続く IF 文でエディター画面の左端をクリックし、ブレークポイントを設定します。



```

BATCHRPT EMPSEQ.cpy BATCHRPT.cbl Program1.cbl Form1
CBL BATCHRPT
MOVE CNTL-MON TO RPT-SELECTION-MM.  

MOVE CNTL-DAY TO RPT-SELECTION-DD.  

MOVE CNTL-YR TO RPT-SELECTION-YYYY.  

WRITE RPT-RECORD FROM RPT-TITLE-3 BEFORE ADVANCING 1 LINE.  

WRITE RPT-RECORD FROM BLANK-LINE BEFORE ADVANCING 1 LINE.  

WRITE RPT-RECORD FROM RPT-COLUMNS BEFORE ADVANCING 1 LINE.  

WRITE RPT-RECORD FROM BLANK-LINE BEFORE ADVANCING 1 LINE.  

1個の参照  

1000-EXIT.  

    EXIT.  

1個の参照  

2000-MAIN-PROCESSING.  

    READ EMP-SEQ-FILE INTO EMP-RECORD-IO-AREA  

        AT END  

            MOVE 'Y' TO EOF-FLAG.  

IF NOT-AT-EOF  

    PERFORM 3000-PROCESS-RECORD THRU 3000-EXIT  

END-IF.

```

メニューより、[デバッガ(D)] > [続行(C)] を選択するか F5 キーを押すと、デバッガーは最初のブレークポイントで実行を中断します。

ウォッチ式の CONTROL-REC の値に制御ファイルから読み込んだレコードが表示されます。

BATCHRPT | EMPSEQ.cpy | **BATCHRPT.cbl** | Program1.cbl | Form1.cbl | Form1.cbl [i]

BATCHRPT

```

MOVE CURR-MIN TO RPT-CURR-MIN.
MOVE CURR-MIN TO RPT-CURR-MIN.
MOVE CURR-SEC TO RPT-CURR-SEC.

WRITE RPT-RECORD FROM RPT-TITLE-1 BEFORE ADVANCING 1 LINE.
WRITE RPT-RECORD FROM RPT-TITLE-2 BEFORE ADVANCING 1 LINE.

*** READ CONTROL CARD FILE TO GET DATE FOR SELECTION CRITERIA.
* IF FILE IS EMPTY, DEFAULT CNTL-DATE TO CURRENT DATE.
*** READ IN-CNTL-CARD INTO CONTROL-REC.

IF CNTL-DATE = SPACES
  MOVE CURR-DATE TO CNTL-DATE
END-IF.

* ACCEPT CNTL-DATE FROM SYSIN.

MOVE CNTL-MON TO RPT-SELECTION-MM.
MOVE CNTL-DAY TO RPT-SELECTION-DD.
MOVE CNTL-YR TO RPT-SELECTION-YYYY.

WRITE RPT-RECORD FROM RPT-TITLE-3 BEFORE ADVANCING 1 LINE.
WRITE RPT-RECORD FROM BLANK-LINE BEFORE ADVANCING 1 LINE.

```

100 % 開問題は見つかりませんでした

ウォッチ 1

名前	値	種類
CONTROL-REC	{長さ=8} : "20000101"	表示 GROUP
EMP-RECORD-IO-AREA	{長さ=60} : .	表示 GROUP

再度、メニューより、[デバッグ(D)] > [続行(C)] を選択するか F5 キーを押すと、デバッガーは 2 番目のブレークポイントで実行を中断します。

ウォッチしている EMP-RECORD-IO-AREA の値に入力ファイルから読み込んだ 1 番目のレコードが表示されます。

BATCHRPT | EMPSEQ.cpy | **BATCHRPT.cbl** | Program1.cbl | Form1.cbl | Form1.cbl [デザイ]

BATCHRPT

```

WRITE RPT-RECORD FROM RPT-COLUMNS BEFORE ADVANCING 1 LINE.
WRITE RPT-RECORD FROM BLANK-LINE BEFORE ADVANCING 1 LINE.

1個の参照
1000-EXIT.
  EXIT.
1個の参照
2000-MAIN-PROCESSING.
  READ EMP-SEQ-FILE INTO EMP-RECORD-IO-AREA
    AT END
      MOVE 'Y' TO EOF-FLAG.

  IF NOT AT EOF
    ! PERFORM 3000-PROCESS-RECORD THRU 3000-EXIT
  END-IF.


```

14件の参照 開問題は見つかりませんでした

ウォッチ 1

名前	値	種類
CONTROL-REC	{長さ=8} : "20000101"	表示 GROUP
EMP-RECORD-IO-AREA	{長さ=60} : "11111113 佐藤 隆 サトウ ル..."	表示 GROUP
EMP-REC	{長さ=60} : "11111113 佐藤 隆 サトウ ル..."	表示 GROUP
EMPREC-SSN	11111113	表示 PIC X(8)
FILLER		表示 PIC X
EMPREC-JNAME1	佐藤	表示 PIC N(5)
EMPREC-JNAME2	隆	表示 PIC N(5)
EMPREC-NAME1	サトウ	表示 PIC X(5)
EMPREC-NAME2	タシ	表示 PIC X(5)
EMPREC-GENDER	M	表示 PIC X
FILLER		表示 PIC X
EMPREC-DIV	営業部	表示 PIC N(5)
EMPREC-DATE-OFF-HIRE	{長さ=8} : "19980401"	表示 GROUP

再度、メニューより、[デバッグ(D)] > [続行(C)] を選択するか F5 キーを押すと、同じブレークポイント位置で停止しますが、2 番目のデータが表示されます。

BATCHRPT | EMPSEQ.cpy | BATCHRPT.cbl | Program1.cbl | Form1.cbl | Form1.cbl [デバ]

CBL BATCHRPT

```
WRITE RPT-RECORD FROM RPT-COLUMNS BEFORE ADVANCING 1 LINE.  
WRITE RPT-RECORD FROM BLANK-LINE BEFORE ADVANCING 1 LINE.
```

1個の参照
1000-EXIT.
 EXIT.
1個の参照
2000-MAIN-PROCESSING.
 READ EMP-SEQ-FILE INTO EMP-RECORD-IO-AREA
 AT END
 MOVE 'Y' TO EOF-FLAG.

 IF NOT AT EOF
 PERFORM 3000-PROCESS-RECORD THRU 3000-EXIT
 ENDIF.

1行未満で空白

問題は見つかりませんでした

ウォッチ 1

名前	値	種類
CONTROL-REC	{長さ=8} : "20000101"	Q 表示 ▾ GROUP
EMP-RECORD-IO-AREA	{長さ=60} : "2222226 鈴木 尚之 及*キナ..."	Q 表示 ▾ GROUP
EMP-REC	{長さ=60} : "2222226 鈴木 尚之 及*キナ..."	Q 表示 ▾ GROUP
EMPREC-SSN	2222226	Q 表示 ▾ PIC X(8)
FILLER		Q 表示 ▾ PIC X
EMPREC-JNAME1	鈴木	Q 表示 ▾ PIC N(5)
EMPREC-JNAME2	尚之	Q 表示 ▾ PIC N(5)
EMPREC-NAME1	及*キ	Q 表示 ▾ PIC X(5)
EMPREC-NAME2	ワ2	Q 表示 ▾ PIC X(5)
EMPREC-GENDER	M	Q 表示 ▾ PIC X
FILLER		Q 表示 ▾ PIC X
EMPREC-DIV	技術部	Q 表示 ▾ PIC N(5)
EMPREC-DATE-OF-HIRE	{長さ=8} : "19981015"	Q 表示 ▾ GROUP

ブレークポイントは、現在設定中の行の左端をクリックすることで解除できます。さきほど設定した2つのブレークポイントを解除してください。

BATCHRPT | EMPSEQ.cpy | **BATCHRPT.cbl** | Program1.cbl | Form1.c

```

CBL BATCHRPT
*** WRITE INT RECORD FROM INT-TITLE-2 BEFORE ADVANCING 1 LINE.
*| READ CONTROL CARD FILE TO GET DATE FOR SELECTION CRITERIA.
*| IF FILE IS EMPTY, DEFAULT CNTL-DATE TO CURRENT DATE.
*** READ IN-CNTL-CARD INTO CONTROL-REC.

IF CNTL-DATE = SPACES
  MOVE CURR-DATE TO CNTL-DATE
END-IF.

ACCEPT CNTL-DATE FROM SYSIN.

MOVE CNTL-MON TO RPT-SELECTION-MM.
MOVE CNTL-DAY TO RPT-SELECTION-DD.
MOVE CNTL-YR TO RPT-SELECTION-YYYY.

WRITE RPT-RECORD FROM RPT-TITLE-3 BEFORE ADVANCING 1 LINE.
WRITE RPT-RECORD FROM BLANK-LINE BEFORE ADVANCING 1 LINE.

WRITE RPT-RECORD FROM RPT-COLUMNS BEFORE ADVANCING 1 LINE.
WRITE RPT-RECORD FROM BLANK-LINE BEFORE ADVANCING 1 LINE.

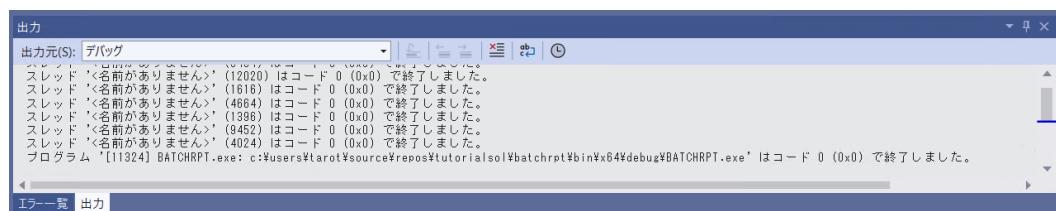
1個の参照
1000-EXIT.
  EXIT.
1個の参照
2000-MAIN-PROCESSING.
  READ EMP-SEQ-FILE INTO EMP-RECORD-IO-AREA
    AT END
      MOVE 'Y' TO EOF-FLAG.

  IIF NOT-AT-EOF
    PERFORM 3000-PROCESS-RECORD THRU 3000-EXIT
  END-IF.

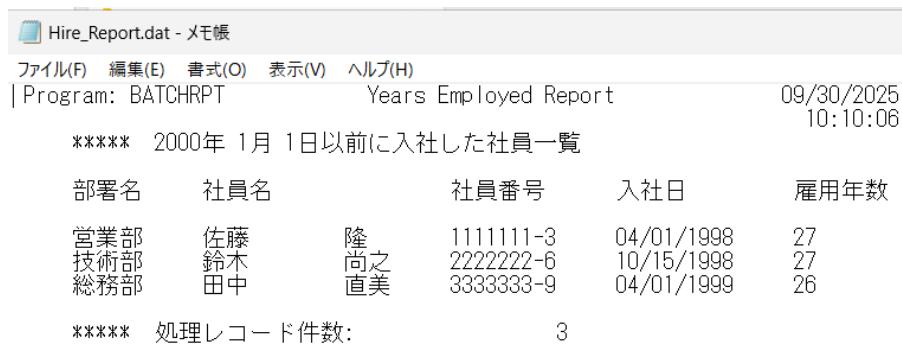
1個の参照
2000-EXIT.
  EXIT.

```

メニューより、[デバッグ(D)] > [続行(C)] を選択するか F5 キーを押して、プログラムを終了します。

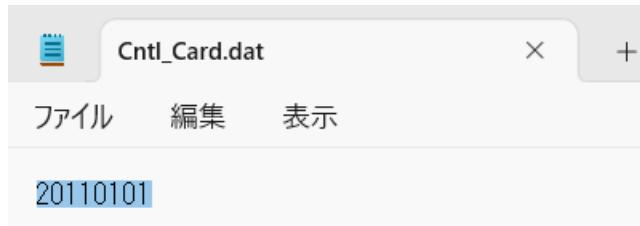


デバッグフォルダ（<3.2 節の 3> 「場所」で指定したフォルダ> ¥TutorialSol¥Tutorial¥bin¥x64¥debug）に Hire_Report.dat ファイルが作成されるので、メモ帳などのエディターでファイルを開き、社員 3 名分のデータが表示されることを確認します。

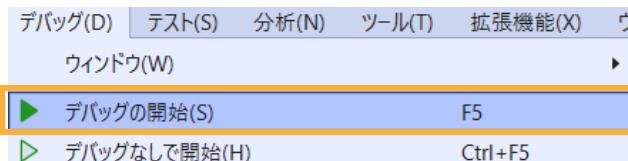


続いて、基準日により結果が変更されることを確認します。

デバッグフォルダ（<3.2 節の 3）「場所」で指定したフォルダ> ¥TutorialSol¥BATCHRPT¥bin¥x64¥debug) に Cntl_Card.dat ファイルの中身を 20110101 に更新します。



メニューより、[デバッグ(D)] > [デバッグの開始(S)] を選択するか F5 キーを押してプログラムを実行します。今は、ブレークポイントが設定されていないため、そのままプログラムは正常終了します。



さきほど確認した Hire_Report.dat ファイルが更新され、2011 年 1 月 1 日以前に入社した社員 9 名分のデータが表示されることを確認します。

Hire_Report.dat - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

Program: BATCHRPT Years Employed Report 09/30/2025
10:12:33

***** 2011年 1月 1日以前に入社した社員一覧

部署名	社員名	社員番号	入社日	雇用年数
営業部	佐藤 隆	1111111-3	04/01/1998	27
技術部	鈴木 尚之	2222222-6	10/15/1998	27
総務部	田中 直美	3333333-9	04/01/1999	26
営業部	山田 洋一	4444444-2	07/01/2000	25
技術部	伊藤 弘子	5555555-5	04/01/2001	24
営業部	木村 貴弘	6666666-8	12/20/2002	23
技術部	中村 慎司	7777777-1	04/01/2003	22
総務部	橋本 悅子	8888888-4	08/05/2004	21
営業部	三井 薫	9999999-7	04/01/2005	20

***** 処理レコード件数: 9

免責事項

ここで紹介したソースコードは、機能説明のためのサンプルであり、製品の一部ではありません。ソースコードが実際に動作するか、御社業務に適合するかなどに関して、一切の保証はございません。ソースコード、説明、その他すべてについて、無謬性は保障されません。

ここで紹介するソースコードの一部、もしくは全部について、弊社に断りなく、御社の内部に組み込み、そのままご利用頂いても構いません。

本ソースコードの一部もしくは全部を二次的著作物に対して引用する場合、著作権法の精神に基づき、適切な扱いを行ってください。