Chapter 1 C++による大規模開発

1.1 デバッガとは

デバッガは、プログラムの動作を解析するためのツールで、プログラムの実行中に問題を特定し、修正することができます。デバッガを使用すると、プログラムを一時停止させることができます。その際、特定の行に到達したときや、特定の条件が満たされたときにプログラムを一時停止するように指示することができます。
例えば、プログラムがエラーで停止している場合、デバッガを使用してエラーが発生した場所を特定することができます。また、プログラムの変数やオブジェクトの値を調べたり、関数がどのように呼び出されたかを追跡したりすることができます。
デバッガは、コードを理解するためのステップバイステップの手順を提供します。プログラムを実行して、それぞれのステップで何が起こっているかを確認することができます。これにより、プログラムの振る舞いをより深く理解することができます。
デバッガは、プログラムの開発に不可欠なツールです。エラーを特定し、修正するためには、プログラムの動作を理解することが必要です。デバッガを使用することで、プログラムの動作をより詳細に理解することができ、開発プロセスを効率的に進めることができます。

1.2 大規模開発とデバッガ

大規模開発において、デバッガは非常に重要なツールとなります。大規模なプログラムを開発する場合、特に複数の人間が開発に関わっているとき、バグが発生する可能性が高くなります。このような大規模な開発において、バグのの発生場所を特定することは困難であり、バグの修正に多くの時間がかかることがあります。
このような場合、デバッガを使用することで、バグが発生した場所を特定することができます。
もし、このような大規模開発でデバッガが使えないとバグの特定と修正にかかる時間が大幅に増加し、プログラム開発の効率が低下することが予想されます。もしデバッガが使えない場合には、代替手段として、ログ出力や単体テストなどを利用することで、バグの特定と修正に時間をかけることなくプログラム開発を進めることができます。ただし、これらの手法はデバッガに比べると効率が低く、開発にかかる時間が増加する可能性があるため、デバッガを使用できる環境を整えることが望ましいといえます。

1.3 Visual Studio(IDE)とデバッガ

Visual Studioは、Microsoftが提供する統合開発環境(IDE)です。統合開発環境（IDE：Integrated Development Environment）は、プログラム開発を行う際に必要とされる様々なツールや機能を一つのソフトウェアパッケージにまとめたものです。主にソフトウェア開発者が使用し、プログラムの開発、編集、コンパイル、実行、テスト、デバッグなどの作業を効率的に行うことができます。

一般的なIDEには、以下のような機能があります。

1.3.1 ソースコードエディター

プログラムを作成するためのテキストエディターで、主要なプログラミング言語に対応しています。また、スペルチェック、コードの自動整形、コードの補完、ファイルの差分の表示などの機能があります。

1.3.2 コンパイラ、インタプリター、デバッガ

プログラムのコンパイル、実行、デバッグを行うためのツールです。これらのツールをIDEで統合することで、より簡単に開発を行うことができます。

1.3.3 ビルドツール、デプロイツール

プログラムのビルドやデプロイメントに必要なツールが統合されています。ビルドツールは、コードのコンパイル、リンク、パッケージ化を自動化するための機能があります。デプロイツールは、アプリケーションを開発環境から実行環境にデプロイするための機能があります。

1.3.4 プロジェクト管理機能

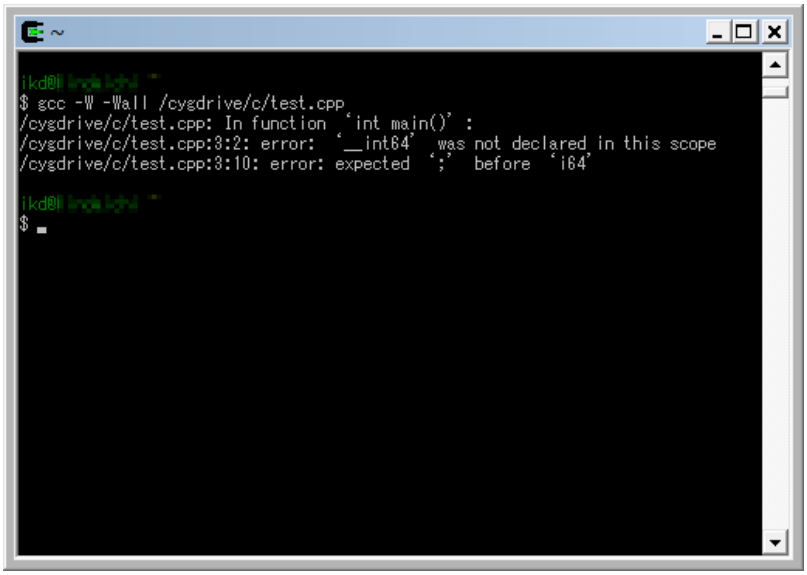
複数のファイルから構成されるプログラムを管理するための機能があります。プロジェクトファイルを作成し、ソースコードファイルやリソースファイルを追加したり、ファイルの結合、参照の解決、依存関係の解決などを自動化することができます。

1.3.5 バージョン管理ツールの統合

ソースコードのバージョン管理を行うためのツールがIDEに統合されている場合があります。これにより、複数人での開発時におけるソースコードの競合や衝突の解決、コードの変更履歴の確認、ロールバックなどが容易になります。

統合環境を利用することで、プログラムの開発プログラムの開発プロセスを効率化することができます。例えば、エディターにコードを書いてからコマンドラインでコンパイルする必要があります。(図1.1)

図1.1



また、コンパイラとエディタなどのツールが別々になっているため、別のウィンドウで作業する必要があったりするため、作業の流れが中断される可能性があります。一方、IDEを使うと、必要な作業がすべて統合されているため、作業の中断を最小限に抑えることができます。

> ただし、ツールを別々で使う場合に比べて、必ずしもIDEの方が優れているというわけではないので、そこな勘違いしないようにしてください。

また、IDEは多くの場合、自動化された作業フローを提供しています。例えば、プログラムのビルドとデプロイの自動化や、テストやデバッグのためのコードスニペットの提供などです。これらの自動化された作業フローにより、開発者はより多くの時間をコードの開発に費やすことができます。

IDEは、開発者が使用する言語やフレームワークに合わせて特化しています。Visual Studioは、.NET FrameworkやC#、Visual Basic、C++に最適化されています。Eclipseは、JavaやAndroidアプリ開発に最適化されています。そのため、開発者は、自分が使用する言語やフレームワークに最適化されたIDEを選ぶことができます。

統合環境は、開発者がプログラム開発に集中できるように設計されています。エディター、コンパイラ、デバッガ、ビルドツール、デプロイツール、プロジェクト管理機能、バージョン管理ツールなど、必要なツールを一つのソフトウェアパッケージにまとめて提供することで、開発者の作業効率を向上させます。また、特定の言語やフレームワークに特化しており、開発者が使用する言語やフレームワークに最適化された機能を提供します。 Visual Studioには、C++、C#、Visual Basicなどのプログラミング言語をサポートする開発ツールが含まれており、プログラムの作成、編集、ビルド、実行、デバッグなどの作業を統合的に行うことができます。

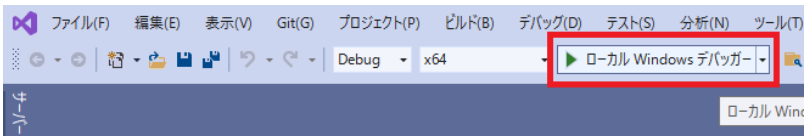
Visual Studioの中でも、デバッグ機能は非常に強力で、多くのデバッグ機能を提供しています。Visual Studioのデバッグ機能を使用することで、ブレークポイントの設定、ステップ実行、ウォッチウィンドウの使用、コールスタックの参照、変数の値の表示など、プログラムのデバッグに必要な機能を簡単に使用することができます。

1.4 Visual Studioでのデバッガの利用(基本)

1.4.1 デバッガありで実行(F5)

デバッガを使うためには、プログラムをデバッガありで実行する必要があります。
「デバッガありでの実行」は図1.2のボタンをクリックするか、ショートカットキーのF5を入力することで行えます。

図1.2



1.4.2 デバッガの停止(shift+F5)

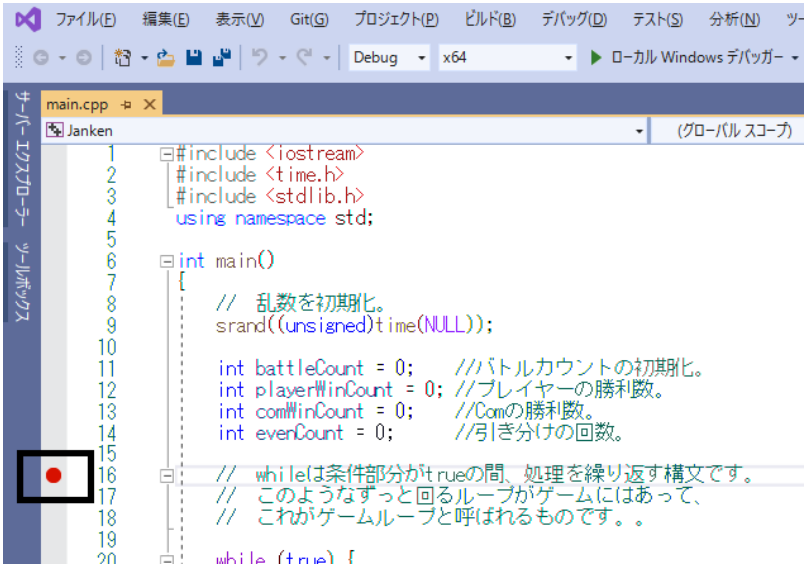
デバッガありで実行開始したプログラムはデバッガの停止ボタンを押すことで停止することができます(図1.3)。
デバッガの停止はショートカットキーのshift+F5でも停止することができます。

図1.3

1.4.3 ブレイクポイント(F9)

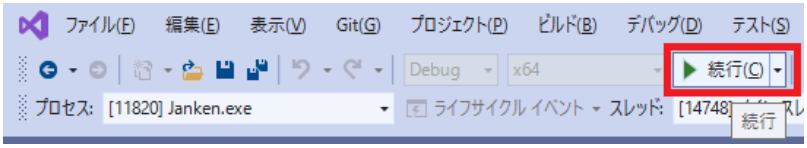
ブレイクポイントを設置することで実行中のプログラムを停止することができます。
プログラムの処理の流れの確認や、後述するウォッチ機能を利用しての変数の値などを行うときに頻繁に利用する機能です。
ブレイクポイントは図1.4のようにプログラムを停止させたい行をマウスでクリックすることで設置することができます。
また、ショートカットキーのF9を入力することでも設置することができます。

図1.4



ブレイクポイントの解除は、解除したいブレイクポイントをマウスでクリックするかF9を入力することで解除することができます。
また、ブレイクポイントで停止したプログラムを再開するには、図1.5の「続行」をクリックするか、ショートカットキーのF5を入力してください。

図1.5



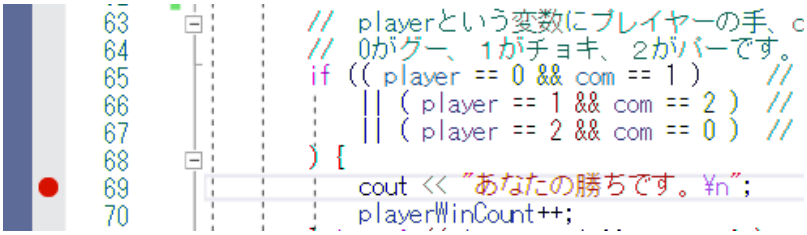
1.4.4【ハンズオン】じゃんけんに勝利したときif文の中にブレイクポイントを設置して見よう

じゃんけんゲームのプログラムでデバッガの機能を利用して、勝利したときのif文の処理が正しく実行されているか確認してみましょう。
Sample_01/Janken.slnを立ち上げてください。

step-1 ブレイクポイントの設置

main.cppの69行目にブレイクポイントを設置してください(図1.6)。

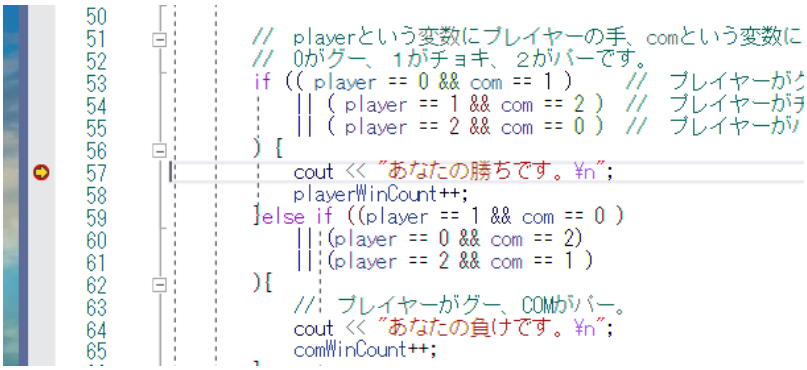
図1.6



step-2 デバッガありで実行

プログラムをデバッガありで実行して、じゃんけんゲームをプレイしてみてください。
ブレイクポイントがうまく設置できていると、じゃんけんに勝利したときに図1.7のようにプログラムが停止するようになります。

図1.7



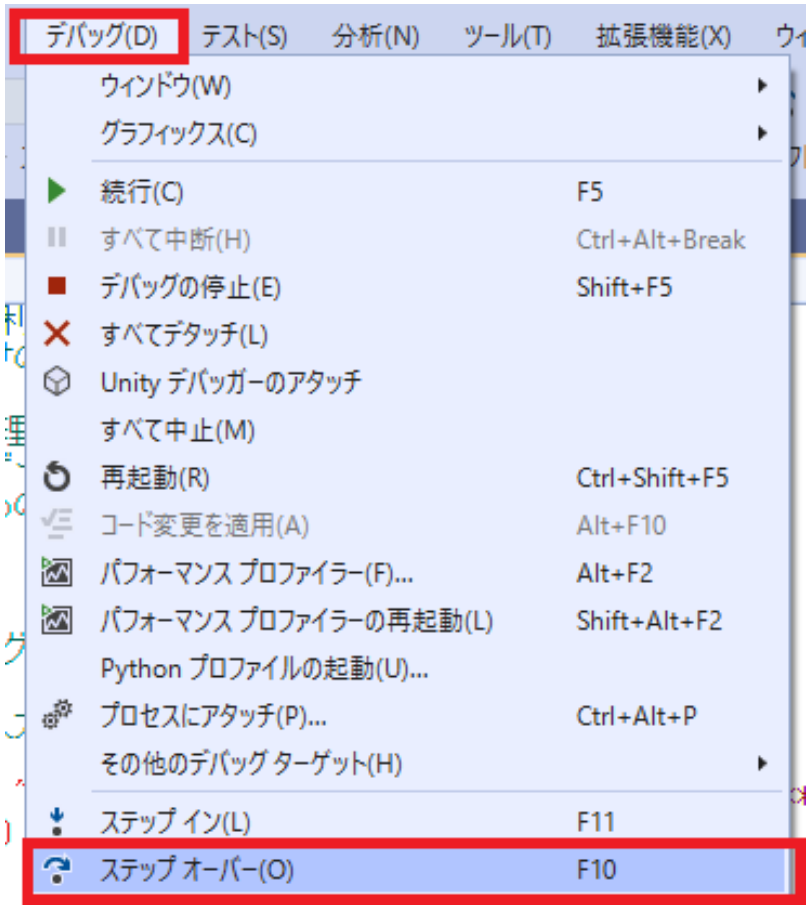
step-3 続行

続行ボタンを押してプログラムを続行してください。

1.4.5 ステップオーバー(F10)

ここからは、プログラムのステップを進めるための機能を見ていきます。
まずはステップオーバーです。
ステップオーバーはブレイクポイントで停止させたプログラムを１行進めることができます。
ステップオーバーはメニューの「デバッグ/ステップオーバー」で実行できます(図1.8)。ショートカットキーのF10でも実行できます。

図1.8



ステップオーバーは関数の中には入らずにステップを進めるため、ステップオーバーと呼ばれます。

1.4.6 ステップイン(F11)

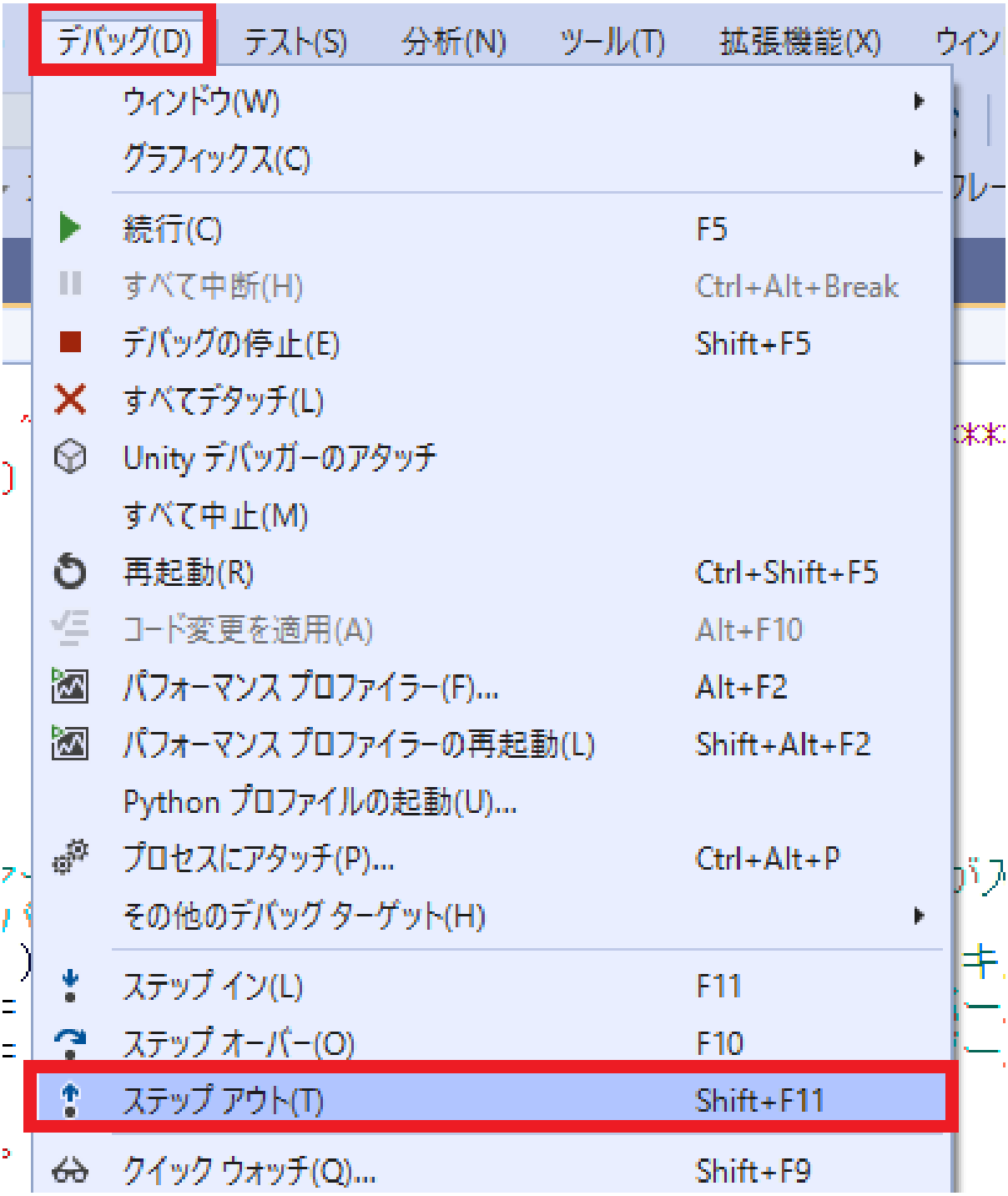
ステップインもプログラムのステップを１行進めるのですが、こちらはステップオーバーとは違い、関数の中に入ることができます。
ステップインはメニューの「デバッグ/ステップイン」で実行できます(図1.9)。
ショートカットキーのF11でも実行できます。

図1.9

1.4.7 ステップアアウト(Shift + F11)

ステップアウトは関数を抜けることができます。
ステップアウトはメニューの「デバッグ/ステップアウト」で実行できます(図1.10)。
ショートカットキーのshift + F11でも実行できます。

図1.10



1.4.8【ハンズオン】ステップオーバー/ステップイン/ステップアウトを試す

では、ハンズオンで各種ステップ実行を試してみましょう。
先ほどのハンズオンでじゃんけんに勝利した際にブレイクポイントを設置していると思いますので、
一旦そこのブレイクポイントでプログラムを停止させてからステップ実行をしていきましょう。
では、まずはプログラムをデバッガありで実行して、じゃんけんゲームをプレイして、69行目のブレイクポイントでプログラムを停止させてください。

step-1 ステップオーバーで61行目までプログラムを進める。

step-2 ステップインでDispComNoTe()の中に入る

step-3 ステップオーバーでDispComNoTe()の処理を23行目まで進める

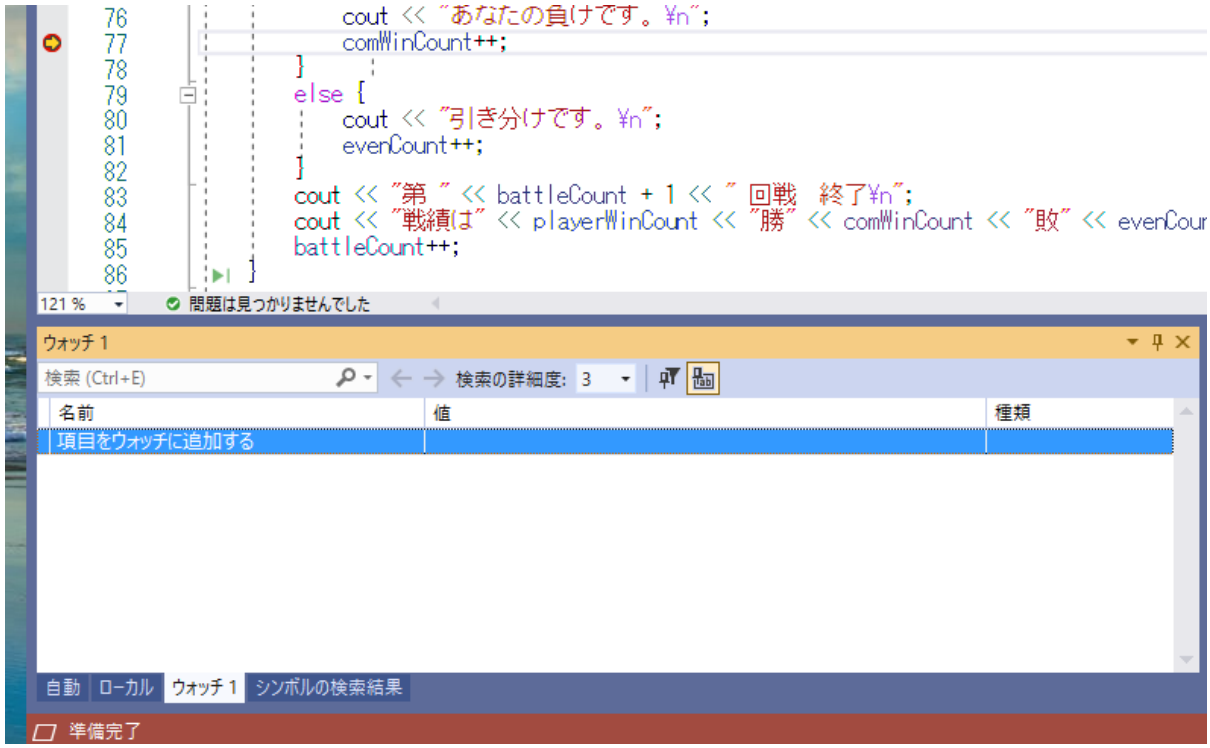
step-4 ステップアウトでDispComNoTe()から抜ける

1.4.9 ウォッチ

プログラム実行中に、プレイヤーの体力など、変数の値がどうなっているのか確認したい場合があります。
このようなときに使える機能がウォッチです。
ウォッチを使うためには、ブレイクポイントを設置してプログラムを停止させる必要があります。
プログラムが停止すると、その時点での変数の値を確認することができます。
ウォッチはメニューの「デバッグ/ウィンドウ/ウォッチ/ウォッチ1」から開くことができます(図1.11)。
ウォッチを開けると図1.12のように画面下部にウォッチ画面が表示されているはずです。

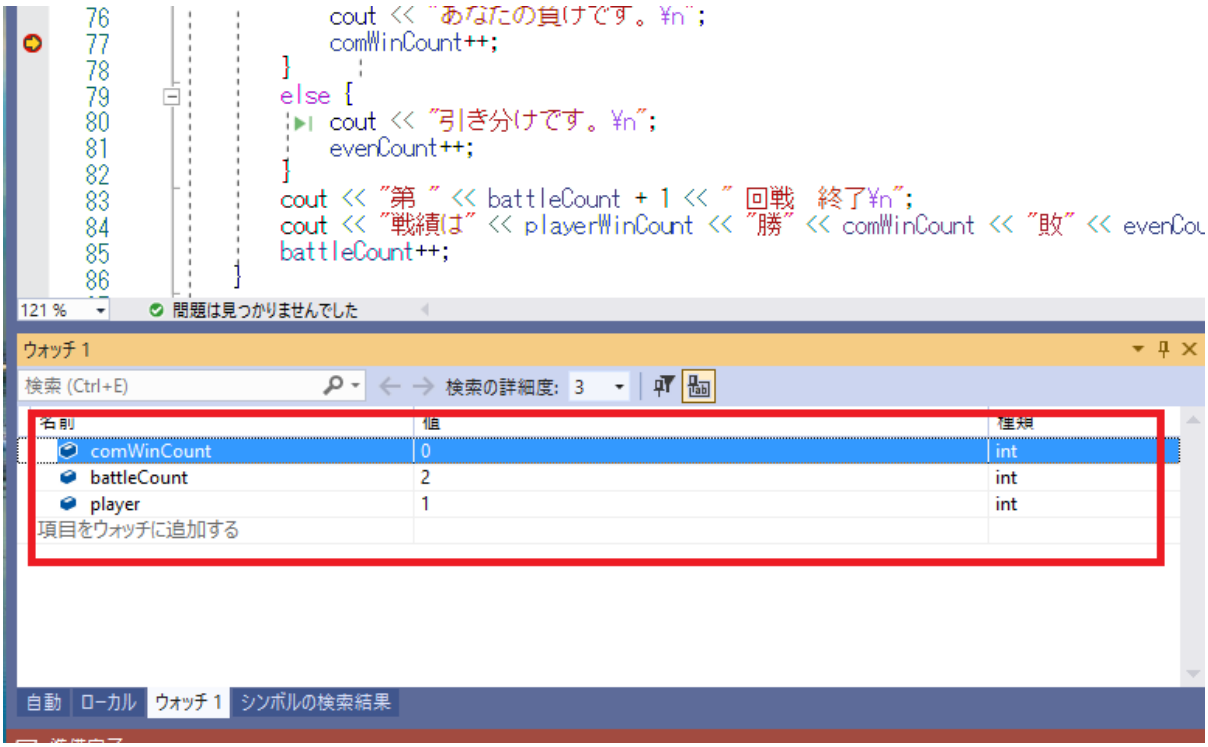
図1.11

図1.12



ウォッチウィンドウ値を確認したい変数を入力 or ドラッグアンドロップすると、変数の値を確認することができます(図1.13)。

図1.13



(注意：ウォッチウィンドウはデバッグありで実行中しか表示されないので注意してください。メニューにも出てきません。)

1.4.10【ハンズオン】ウォッチを使ってみる

では、ウォッチを使ってみましょう。
また、じゃんけんゲームをデバッガありで起動してプレイしてください。
するとじゃんけんで勝利したときにプログラムがブレイクポイントで停止すると思いますので、停止したら変数player、comをウォッチに追加して値を確認してください。

1.1.11 評価テスト-1

次の評価テストを行いなさい。

評価テストへジャンプ

1.1.12 実習課題-1

Question_01のじゃんけんゲームはいくつかプログラムの間違い、バグがあります。
デバッガの機能を活用して、バグの原因を見つけ出してバグを修正しなさい。