

vc-2. Visual Studio C++ のデ バッガー

(Visual Studio C++ の機能と操作演習,
全5回)

<https://www.kkaneko.jp/cc/vc/index.html>

金子邦彦



目次



2-1. デバッガーの機能

2-2. ステップオーバー機能

2-3. 変数の変化, プログラム実行の流れ

2-1 デバッガの機能

2-1 デバッガー

デバッガーとは、プログラムの不具合（バグ）の発見や修正を支援するソフトウェアのこと

※ **バグ**を自動的に取り除いてくれるわけではない

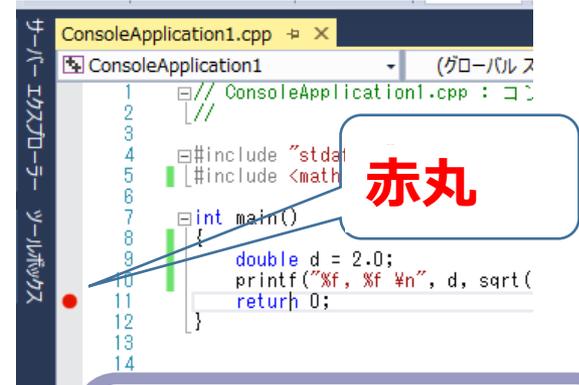
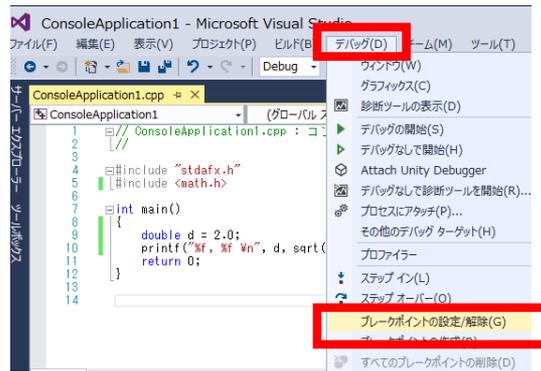
主な機能

- ◆ **ブレークポイント**機能： プログラムの実行中断
- ◆ **トレース**機能： プログラム実行中に**変数**の値などを表示

Visual Studio 2015 でのブレークポイント設定手順



```
1 // ConsoleApplication1.cpp : コンソールアプリケーションのソースファイル
2 //
3
4 #include "stdafx.h"
5 #include <math.h>
6
7 int main()
8 {
9     double d = 2.0;
10    printf("%f, %f\n", d, sqrt(d));
11    return 0;
12 }
13
14
```



① 「return 0;」の行をマウスでクリック

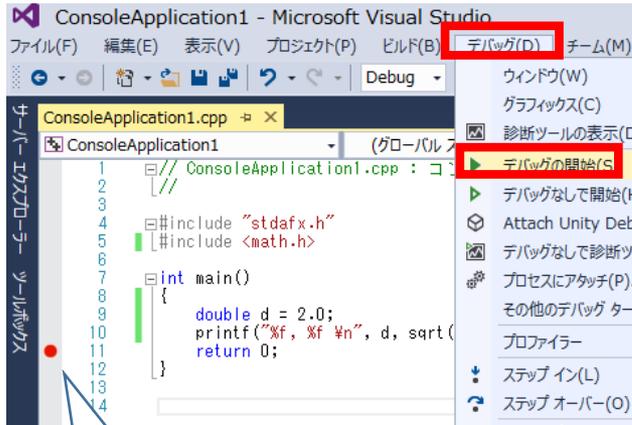
② 「デバッグ」→「ブレークポイントの設定/解除」

③ ブレークポイントが設定されるので確認

ブレークポイントとは、プログラムの実行を中断させたい行。複数設定可

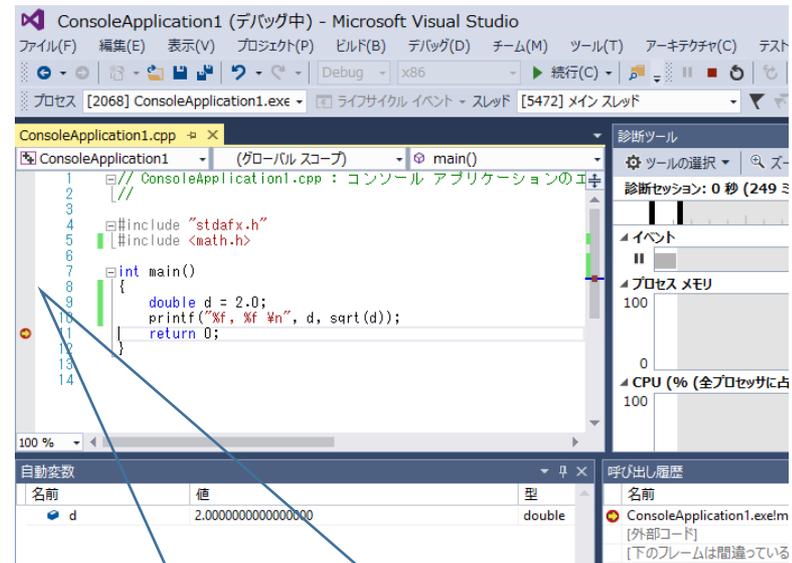
ブレークポイントを解除したいときは赤丸をクリック
※ 同様の操作で、ブレークポイントの設定もできる

Visual Studio 2015 でのデバッガー一起動手順

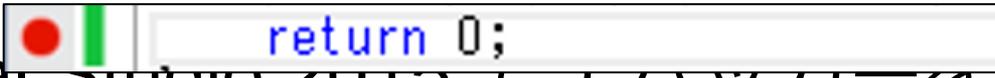


① 「デバッグ」
→ 「デバッグの開始」

ブレークポイントが設定された状態で、デバッガーを起動しようとしている



形が変化！ これは、プログラムの実行が、ここで中断していることを示す

- Visual Studio 2015 で「return 0;」の行にブレークポイントを設定しなさい
-  Visual Studio 2015 でフロッグを起動しなさい。
「return 0;」の行で、実行が中断することを確認しなさい
- あとで使うので、中断したままにしておくこと

2-2 ステップ実行機能

2-1 ステップ実行機能

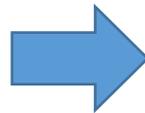
ステップ実行機能を使うことで、

- 変数の値の変化

※ 変数名と値の対応表

- メモリ中のデータの変化 ※ ダンプリスト形式

z	0
x	0
y	0



z	0
x	3
y	0

変数 **x** の値が **0** から **3** に変化
〈ステップ実行機能で表示できる〉

ステップ実行の種類

- ステップオーバー F10キー
 - プログラム実行を1行進める。関数の中に入らない
- ステップイン F11キー
 - プログラム実行を1行進める。関数の中に入る
- ステップアウト Shift と F1 1 キー同時押し
 - 関数の外に出る

※ 「関数」は、
プログラミング言語の関数のこと

Visual C++ のソースファイル例



```
1 // ConsoleApplication2.cpp :
2 //
3
4 #include "stdafx.h"
5
6
7 int main()
8 {
9     static int x, y, z;
10    x = 3;
11    y = 4;
12    z = x + y;
13    return 0;
14 }
15
```

← 4行追加

自動生成されたプログラムを活用（4行追加するだけ）

Visual C++ のソースファイル例



```
1 // ConsoleApplication2.cpp :
2 //
3
4 #include "stdafx.h"
5
6
7 int main()
8 {
9     static int x, y, z;
10    x = 3;
11    y = 4;
12    z = x + y;
13    return 0;
14 }
15
```

変数 **x** に値 **3** をセット
変数 **y** に値 **4** をセット

x と **y** の値を足して、
結果を **z** に格納

※ 「**static** int x, y, z;」の「**static**」は、変数 x, y, z のメモリへの格納法をコントロールするキーワード

変数の値の変化



```
x = 3;
```

この行で、**x** の値は **3** に変化

```
y = 4;
```

この行で、**y** の値は **4** に変化

```
z = x + y;
```

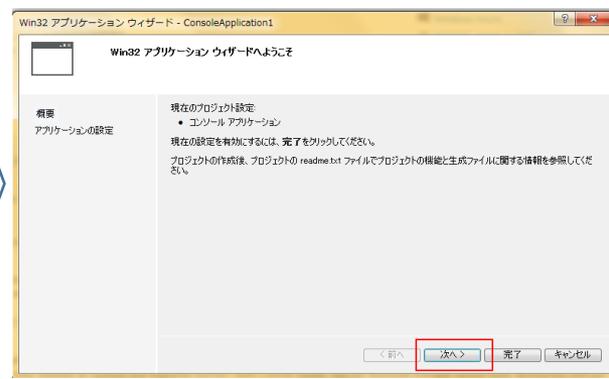
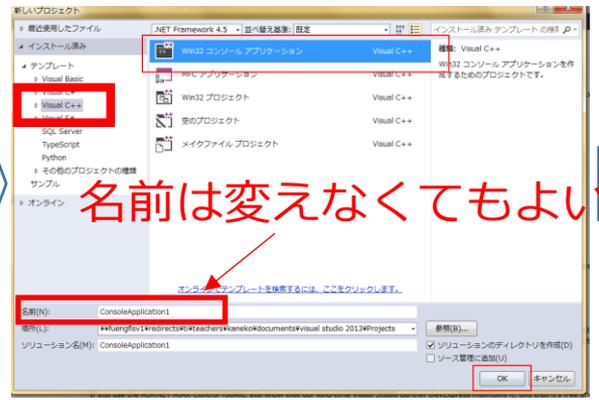
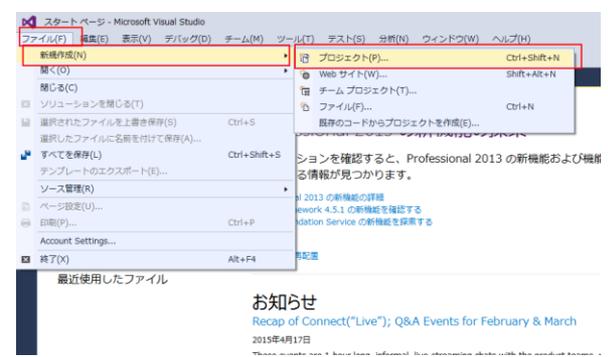
この行で、**z** の値は **7** に変化

ステップオーバー機能

- Visual Studio 2015 を起動しなさい
- Visual Studio 2015 で、Win32 コンソールアプリケーション用プロジェクトを新規作成しなさい

プロジェクトの「名前」は何でもよい

Visual Studio 2015 で Win 32 コンソールアプリケーション用プロジェクトの新規作成 (1/2)

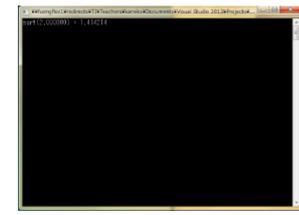


① 「ファイル」 → 「新規作成」 → 「プロジェクト」

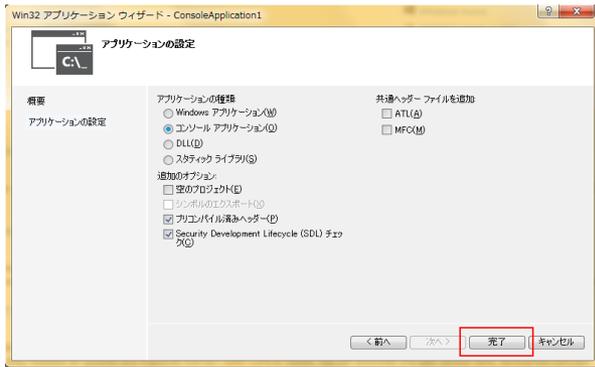
② 「Visual C++」 → 「Win32コンソールアプリケーション」 → 「OK」

③ 「次へ」
※ 次ページに続く

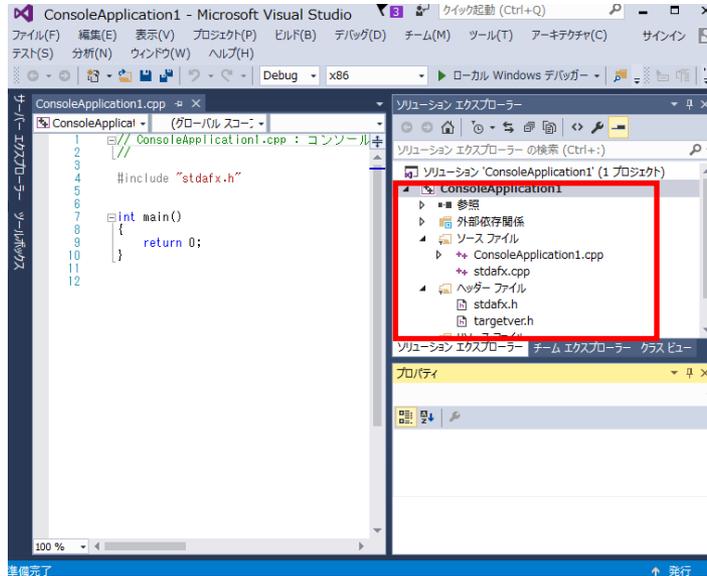
Win32コンソールアプリケーションとは、起動すると、**Win32コンソール**（例えば右図）が開くアプリのこと



Visual Studio 2015 で Win 32 コンソールアプリケーション用プロジェクトの新規作成 (2/2)



④ 「完了」をクリック

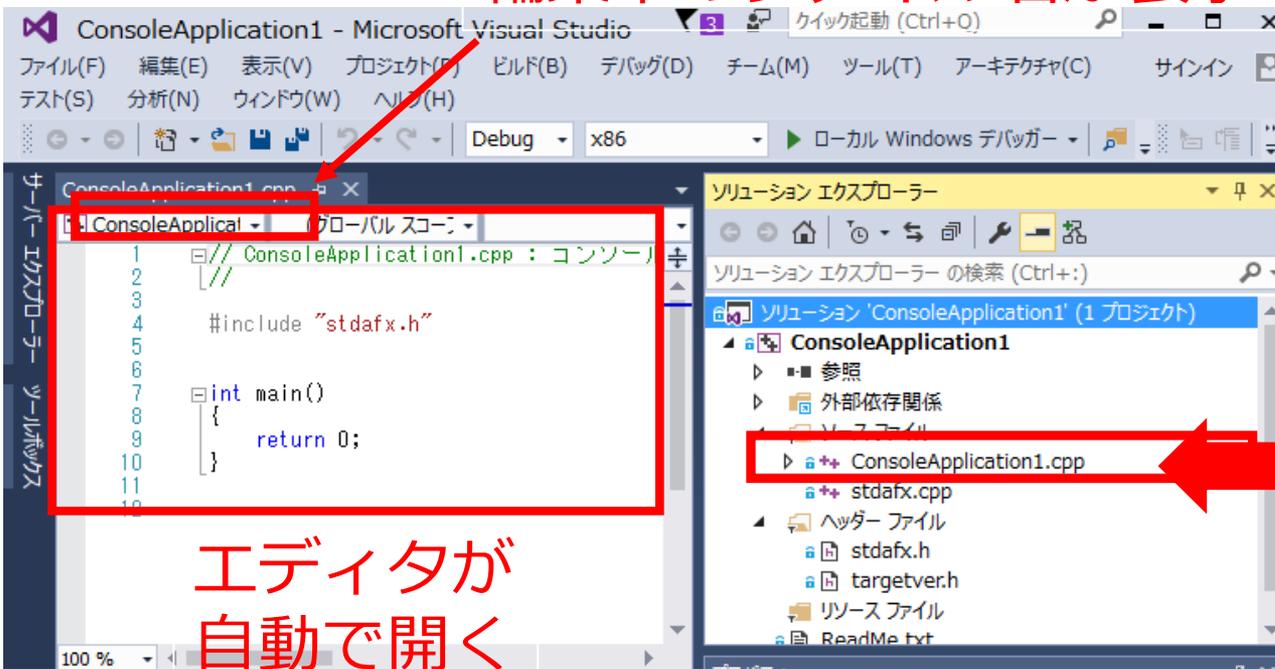


⑤ プロジェクトが新規作成されるので確認

Visual Studio 2015 の Win 32 コンソールアプリケーション用プロジェクト



編集集中のファイル名が表示



エディタが
自動で開く

ソースファイルを
クリックすると

※ Win32 コンソールアプリケーション用プロジェクトを新規作成したとき、エディタが自動で開くので、そのまま使ってよい

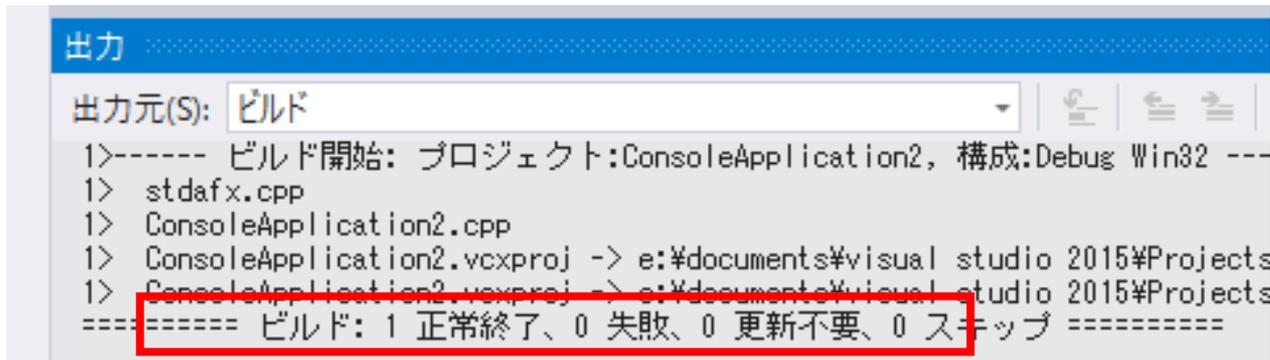
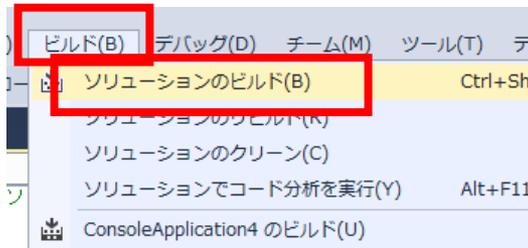
- Visual Studio 2015 のエディタを使って、ソースファイルを編集しなさい

```
1 // ConsoleApplication2.cpp :  
2 //  
3  
4 #include "stdafx.h"  
5  
6  
7 int main()  
8 {  
9     static int x, y, z;  
10    x = 3;  
11    y = 4;  
12    z = x + y;  
13    return 0;  
14 }  
15
```

← 4行追加

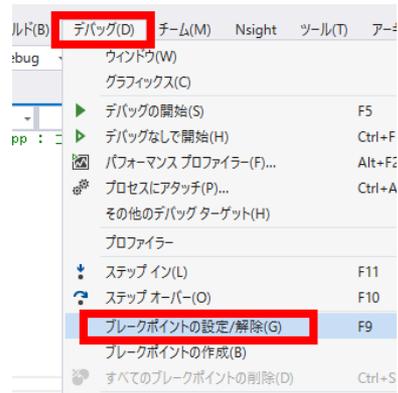
- ビルドしなさい。ビルドのあと「1 正常終了、0 失敗」の表示を確認しなさい

→ 表示されなければ、プログラムのミスを自分で確認し、修正して、ビルドをやり直す



- Visual Studio 2015 で「x = 3;」の行に、ブレークポイントを設定しなさい

```
4 #include <stdio.h>
5
6
7
8 int main()
9 {
10     static int x, y, z;
11     x = 3;
12     y = 4;
13     z = x + y;
14     return 0;
15 }
```



① 「x = 3;」の行を
マウスでクリック

② 「デバッグ」→
「ブレークポイントの
設定/解除」

```
4 #include <stdio.h>
5
6
7
8 int main()
9 {
10     static int x, y, z;
11     x = 3;
12     y = 4;
13     z = x + y;
14     return 0;
15 }
16
```



③ ブレークポイントが設定
されるので確認
赤丸がブレークポイントの印

- Visual Studio 2015 で、デバッガーを起動しなさい。

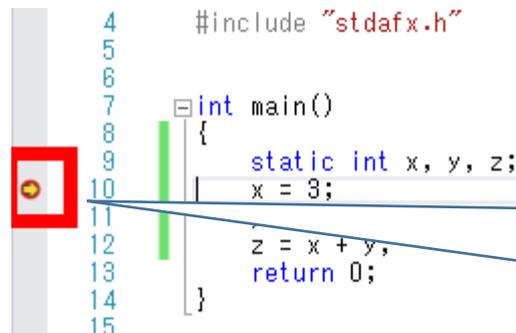


「デバッグ」
→ 「デバッグの開始」

- 「x = 3;」 の行で、実行が中断することを確認しなさい

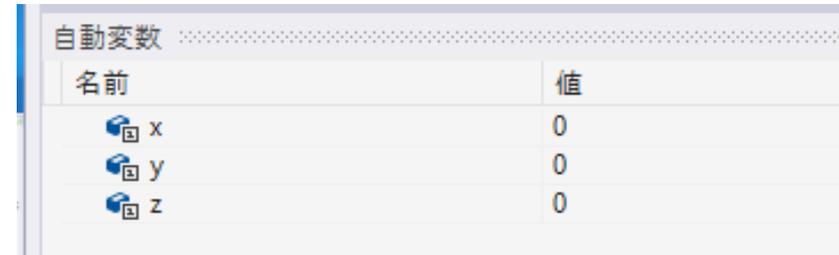
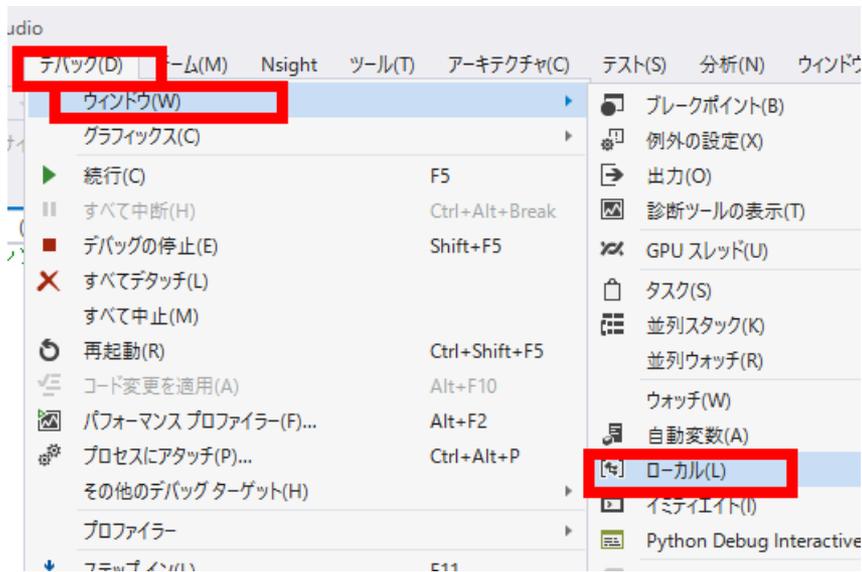
あとで使うので、中断したままにしておくこと

```
4 #include "stdafx.h"
5
6
7 int main()
8 {
9     static int x, y, z;
10    x = 3;
11
12    z = x + y,
13    return 0;
14 }
15
```



「x = 3;」 の行で実行が
中断している

- 「x = 3;」 の行で、実行が中断した状態で、変数の値を表示させなさい。手順は次の通り。



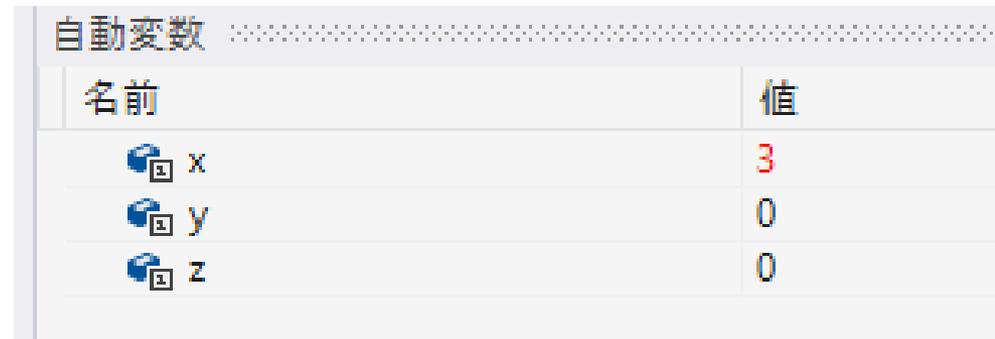
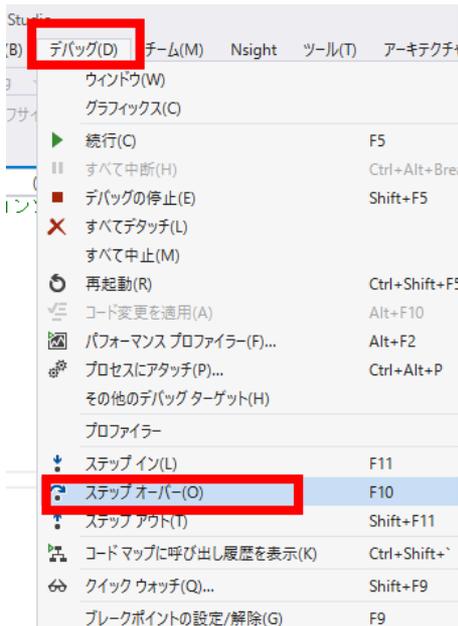
変数 x, y, z の値はすべて 0

② 変数名と値の対応表が表示される

① 「デバッグ」
→ 「ウィンドウ」
→ 「ローカル」

「x = 3;」 は未実行であることを確認！

- 1 回だけステップオーバーの操作を行い、変数 x の値が 3 に変化することを確認しなさい。



The 'Automatic Variables' window shows a table with two columns: '名前' (Name) and '値' (Value). The variable 'x' is highlighted in red, and its value is 3. The other variables 'y' and 'z' have values of 0.

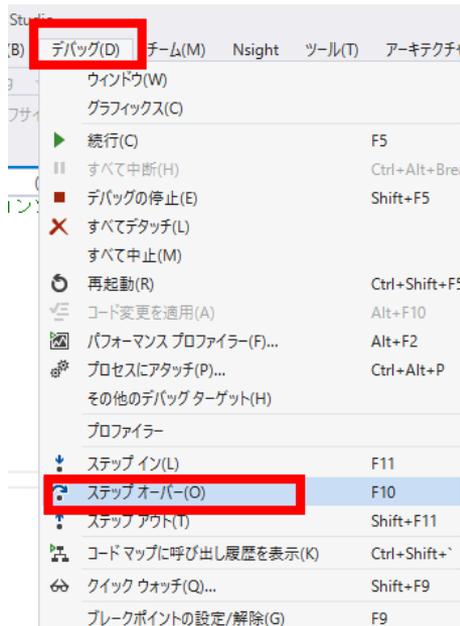
名前	値
x	3
y	0
z	0

変数 x の値が 3 に変化！

「デバッグ」
→ 「ステップオーバー」
(あるいは F10 キー)

「 $x = 3;$ 」が実行された

- もう 1 回だけステップオーバーの操作を行い、変数 y の値が 4 に変化することを確認しなさい。



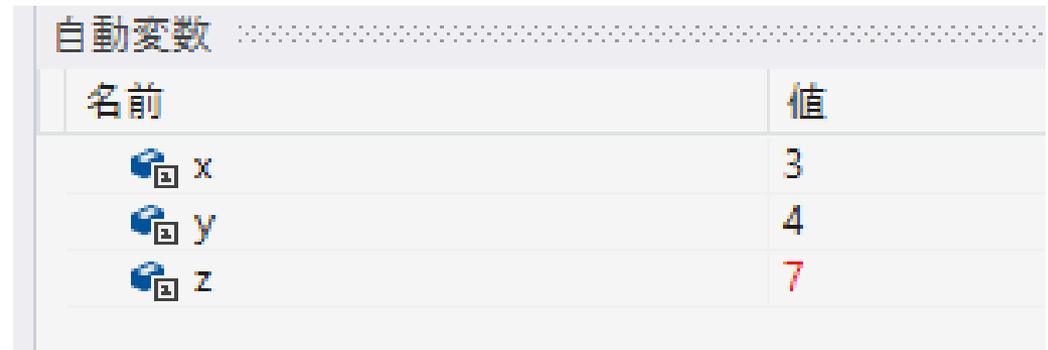
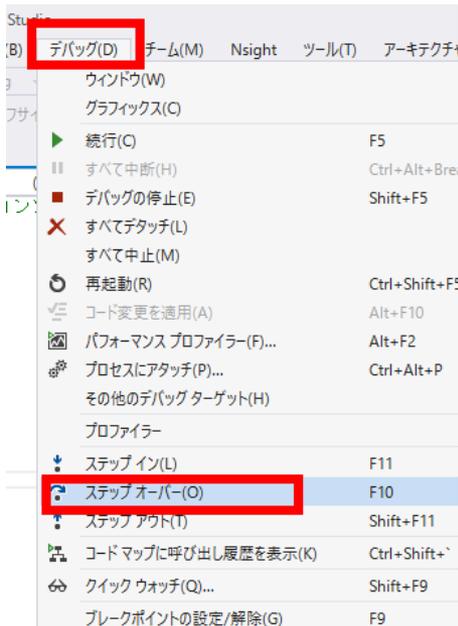
自動変数	
名前	値
x	3
y	4
z	0

変数 y の値が 4 に変化！

「デバッグ」
→ 「ステップオーバー」
(あるいは F10 キー)

「 $y = 4;$ 」が実行された

- もう 1 回だけステップオーバーの操作を行い、変数 z の値が 7 に変化することを確認しなさい。



The image shows the '自動変数' (Automatic Variables) window in Visual Studio. It contains a table with two columns: '名前' (Name) and '値' (Value). The table lists three variables: x with value 3, y with value 4, and z with value 7. The variable z and its value 7 are highlighted in red.

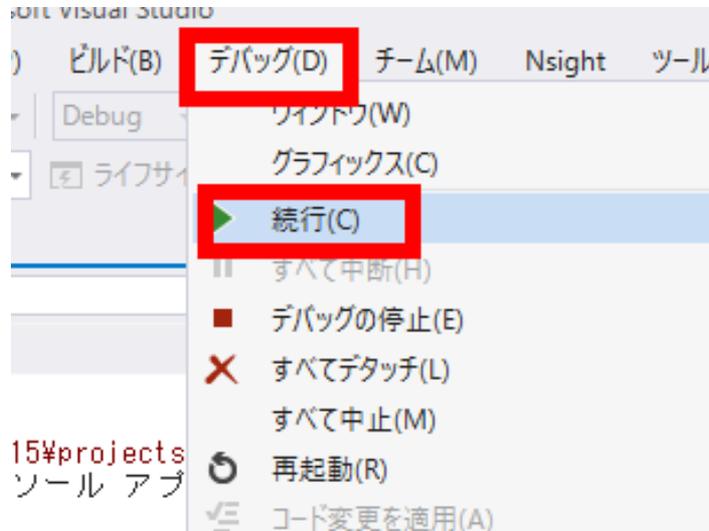
名前	値
x	3
y	4
z	7

変数 z の値が 7 に変化！

「デバッグ」
→ 「ステップオーバー」
(あるいは F10 キー)

「 $z = x + y;$ 」が実行された

- 最後に、プログラム実行の続行の操作を行いなさい。これで、デバッガーが終了する。



「デバッグ」
→ 「続行」

2-3 変数の変化, プログラム 実行の流れ

繰り返しの例



y	x
36	3
60	5
24	2
12	1
24	2
24	2

Visual C++ のプログラム

```
static int x[6] = { 3, 5, 2, 1, 2, 2 };
static int y[6];
int i;
for (i = 0; i < 6; i++) {
    y[i] = x[i] * 12;
}
```

繰り返す処理

i の値は $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$
と変化し, 全部済んだら終わる

- Visual Studio 2015 を起動しなさい
- Visual Studio 2015 で、Win32 コンソールアプリケーション用プロジェクトを新規作成しなさい

プロジェクトの「名前」は何でもよい

- Visual Studio 2015 のエディタを使って、ソースファイルを編集しなさい

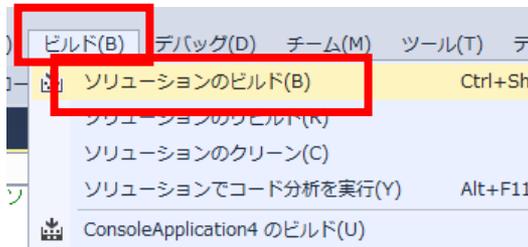
```
int main()  
{
```

```
    static int x[6] = { 3, 5, 2, 1, 2, 2, };  
    static int y[6];  
    int i;  
    for (i = 0; i < 6; i++) {  
        y[i] = x[i] * 12;  
    }
```

6行追加

- ビルドしなさい。ビルドのあと「1 正常終了、0 失敗」の表示を確認しなさい

→ 表示されなければ、プログラムのミスを自分で確認し、修正して、ビルドをやり直す



```
出力
出力元(S): ビルド
1>----- ビルド開始: プロジェクト:ConsoleApplication2, 構成:Debug Win32 ---
1> stdafx.cpp
1> ConsoleApplication2.cpp
1> ConsoleApplication2.vcxproj -> e:\documents\visual studio 2015\Projects
1> ConsoleApplication2.vcxproj -> e:\documents\visual studio 2015\Projects
====
==== ビルド: 1 正常終了、0 失敗、0 更新不要、0 スキップ =====
```

- Visual Studio 2015 で「for (i = 0; ...」の行に、ブレークポイントを設定しなさい



```
7 int main()
8 {
9     static int x[8] = { 3, 5, 2, 1, 2, 2, };
10    static int y[8];
11    int i;
12    for (i = 0; i < 8; i++) {
13        y[i] = x[i] * 12;
14    }
15 }
16
```

デバッグ(D) | ーム(M) | ツール(T) | テスト(S) | 分析(A)

- ウィンドウ(W)
- グラフィックス
- ▶ デバッグ開始(S) F5
- ▶ デバッグなしで開始(H) Ctrl+F5
- ▶ プロセスにアタッチ(P)...
- 例外(X)...
- パフォーマンスと診断(F)
- ALT+F2
- ▶ ステップイン(I) F11
- ▶ ステップオーバー(O) F10
- ▶ ブレークポイントの設定/解除(G) F9
- ブレークポイントの作成(B)
- すべてのブレークポイントの削除(D) Ctrl+Shift+F
- すべてのデータヒントをクリア(A)
- データヒントのエクスポート(X)

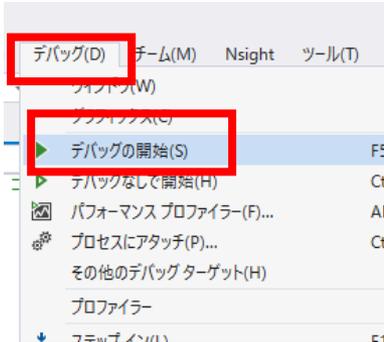
```
6
7
8
9
10 int main()
11 {
12     static int x[8] = { 3, 5, 2, 1, 2, 2, };
13     static int y[8];
14     int i;
15     for (i = 0; i < 8; i++) {
16         y[i] = x[i] * 12;
17     }
18 }
```

① 「for (i = 0;」の行をマウスでクリック

② 「デバッグ」 → 「ブレークポイントの設定/解除」

③ ブレークポイントが設定されるので確認。赤丸がブレークポイントの印

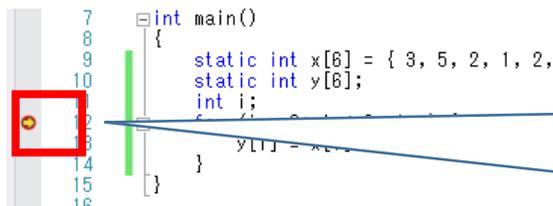
- Visual Studio 2015 で、デバッガーを起動しなさい。



「デバッグ」
→ 「デバッグ開始」

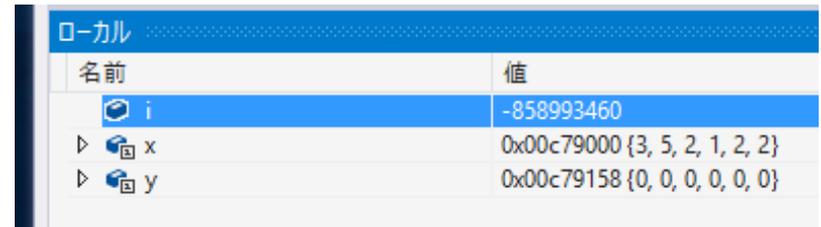
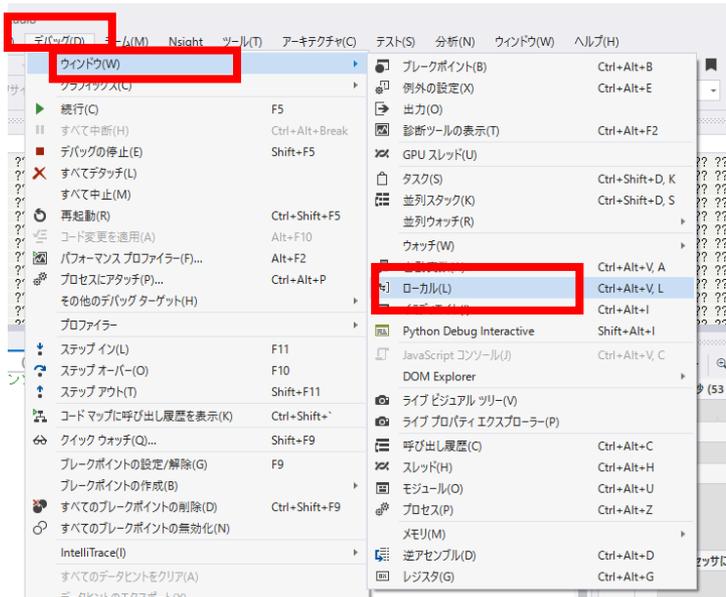
- 「for (i = 0; ...」 の行で、実行が中断することを確認しなさい

あとで使うので、中断したままにしておくこと



「for (i = 0; ...」 の行で実行が
中断している

- 「for (i = 0; ...)」 の行で、実行が中断した状態で、変数の値を表示させなさい。手順は次の通り。



名前	値
i	-858993460
x	0x00c79000 {3, 5, 2, 1, 2, 2}
y	0x00c79158 {0, 0, 0, 0, 0}

② 変数名と値の対応表が表示される

① 「デバッグ」
→ 「ウィンドウ」
→ 「ローカル」

※ 次ページに拡大図

ローカル

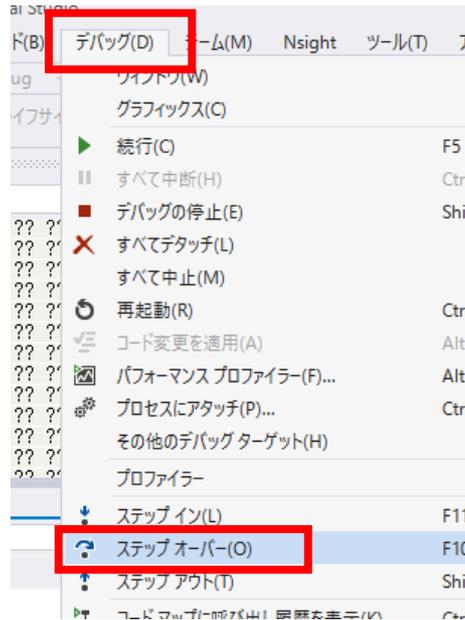
名前

変数 i の値は、変な値になっているはず

	i	-858993460
▷ 	x	0x00c79000 {3, 5, 2, 1, 2, 2}
▷ 	y	0x00c79158 {0, 0, 0, 0, 0, 0}

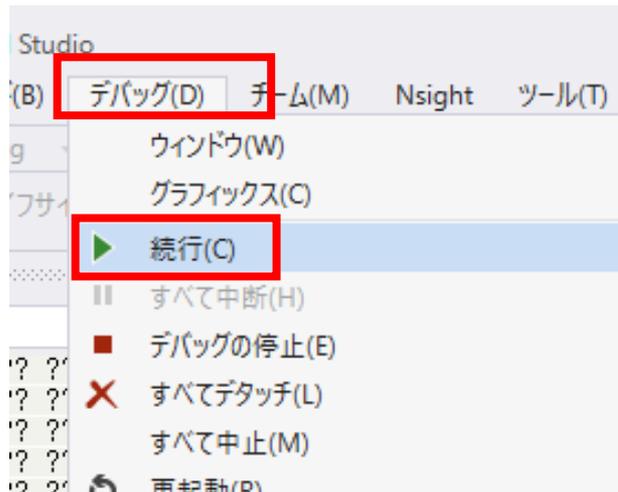
「 $y[i] = x[i] * 12;$ 」は
未実行であることを確認！

- ステップオーバーの操作を1回ずつ行いながら、実行の流れを確認しなさい。



「デバッグ」
→ 「ステップオーバー」
(あるいは F10 キー)

- 最後に、プログラム実行の再開の操作を行いなさい。これで、デバッガーが終了する。



「デバッグ」
→ 「**続行**」

- Visual Studio 2015 を起動しなさい
- Visual Studio 2015 で、Win32 コンソールアプリケーション用プロジェクトを新規作成しなさい

プロジェクトの「名前」は何でもよい

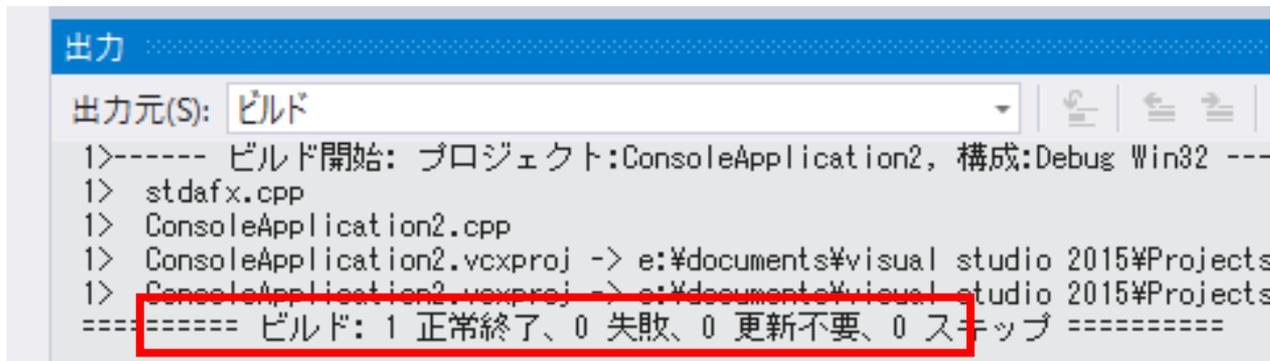
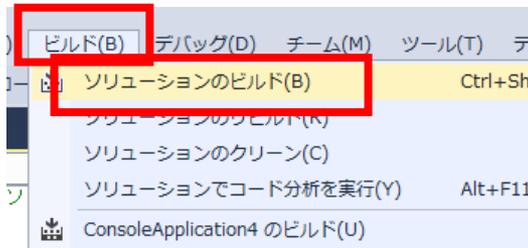
- Visual Studio 2015 のエディタを使って、ソースファイルを編集しなさい

```
int main()  
{  
    int s, i;  
    s = 0;  
    for (i = 1; i <= 3; i++) {  
        s = s + i;  
    }  
    printf("%d\n", s);  
    return 0;  
}
```

← 追加

- ビルドしなさい。ビルドのあと「1 正常終了、0 失敗」の表示を確認しなさい

→ 表示されなければ、プログラムのミスを自分で確認し、修正して、ビルドをやり直す



- Visual Studio 2015 で「s = 0;」の行に、ブレークポイントを設定しなさい

```
int main()
{
    int s, i;
    s = 0;
    for (i = 1; i <= 3; i++) {
        s = s + i;
    }
    printf("%d\n", s);
    return 0;
}
```



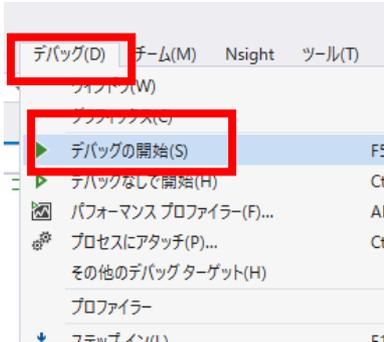
```
int main()
{
    int s, i;
    s = 0;
    for (i = 1; i <= 3;
        s = s + i;
    }
    printf("%d\n", s);
    return 0;
}
```

① 「s = 0;」の行を
マウスでクリック

② 「デバッグ」→
「ブレークポイント
の設定/解除」

③ ブレークポイントが
設定されるので確認。
赤丸がブレークポイン
トの印

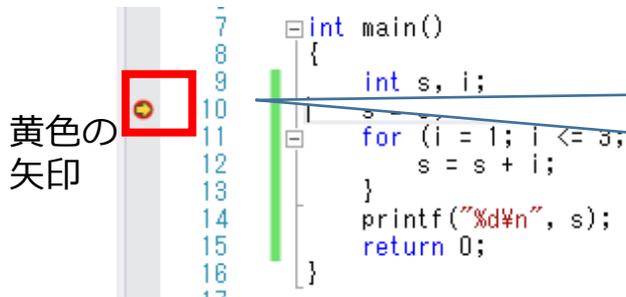
- Visual Studio 2015 で、デバッガーを起動しなさい。



「デバッグ」
→ 「デバッグの開始」

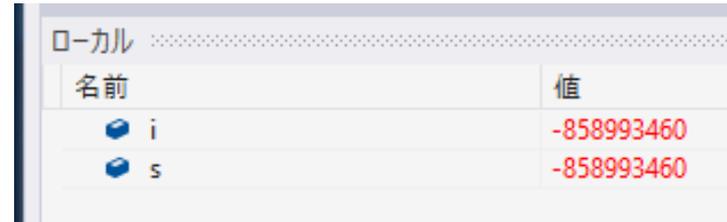
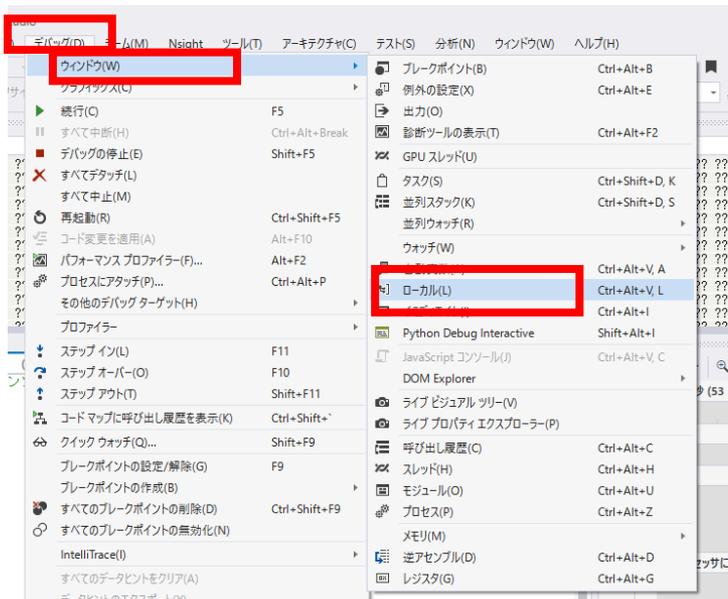
- 「s = 0;」 の行で、実行が中断することを確認しなさい

あとで使うので、中断したままにしておくこと



「s = 0;」 の行で実行が
中断している

- 「s = 0;」の行で、実行が中断した状態で、変数の値を表示させなさい。手順は次の通り。



The image shows the 'Local' window in Visual Studio Code. It displays a table with two columns: '名前' (Name) and '値' (Value). The table contains two rows: one for variable 'i' with value '-858993460', and one for variable 's' with value '-858993460'.

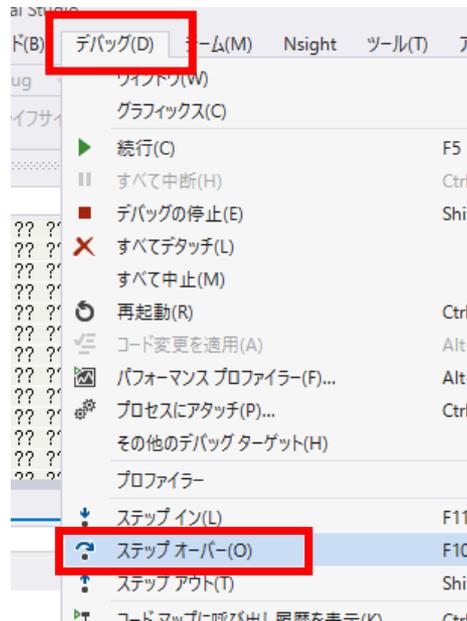
名前	値
i	-858993460
s	-858993460

② 変数名と値の対応表が表示される

- ① 「デバッグ」
→ 「ウィンドウ」
→ 「ローカル」

変数の値がおかしい
→ これは**正しい動作**。
今の時点では、
「s = 0;」は**未実行**であることを確認！

- ステップオーバーの操作を1回ずつ行いながら、実行の流れと、変数 s , i の値の変化を確認しなさい。



「デバッグ」
→ 「ステップオーバー」
(あるいは F10 キー)

ローカル	
名前	値
i	-858993460
s	-858993460



ローカル	
名前	値
i	-858993460
s	0

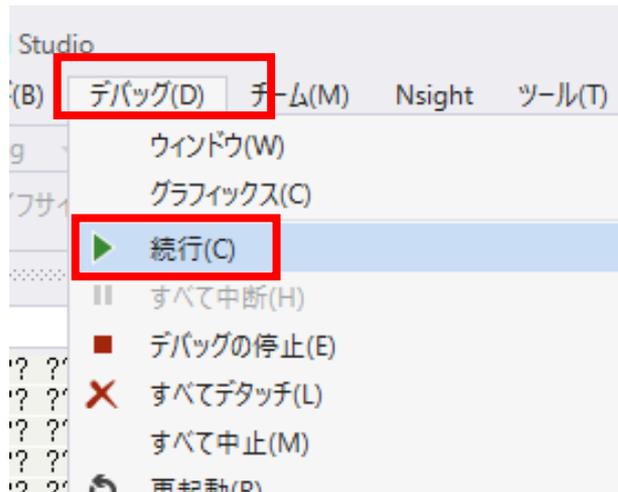


ローカル	
名前	値
i	1
s	0



さらに続ける

- 最後に、プログラム実行の再開の操作を行いなさい。これで、デバッガーが終了する。



「デバッグ」
→ 「**続行**」