

ca-13. サブルーチン

(コンピュータ・アーキテクチャ演習)

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

金子邦彦



アウトライン



13-1 サブルーチン

13-2 サブルーチン呼び出しと復帰

13-3 演習

13-1 サブルーチン

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

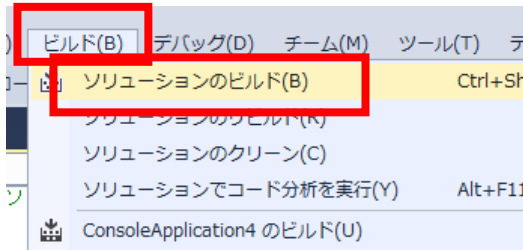
プロジェクトの「名前」は何でもよい

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

```
4      #include "stdafx.h"
5
6      double foo(double a)
7      {
8          return a * 1.08;
9      }
10
11     int main()
12     {
13         double p;
14         p = 120;
15         printf("%f¥n", foo(p));
16         p = 200;
17         printf("%f¥n", foo(p));
18         return 0;
19     }
```

- 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

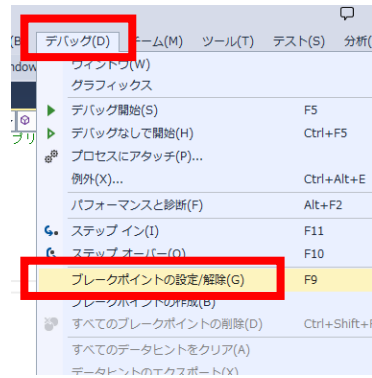
→ 表示されなければ, プログラムのミスを自分で確認し, 修正して, ビルドをやり直す



```
出力
出力元(S): ビルド
1>----- ビルド開始: プロジェクト:ConsoleApplication6, 構成:Debug Win32 -----
1> stdafx.cpp
1> ConsoleApplication6.cpp
1> ConsoleApplication6.vcxproj -> e:\documents\visual studio 2015\Projects\
1> ConsoleApplication6.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で「return a * 1.08;」の行に、ブレークポイントを設定しなさい

```
4 #include "stdafx.h"
5
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f¥n", foo(p));
16     p = 200;
17     printf("%f¥n", foo(p));
18     return 0;
19 }
```



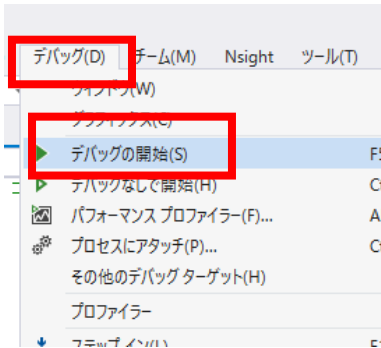
```
4 #include "stdafx.h"
5
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f¥n", foo(p));
16     p = 200;
17     printf("%f¥n", foo(p));
18     return 0;
19 }
```

① 「return a * 1.08;」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で、デバッガーを起動しなさい。



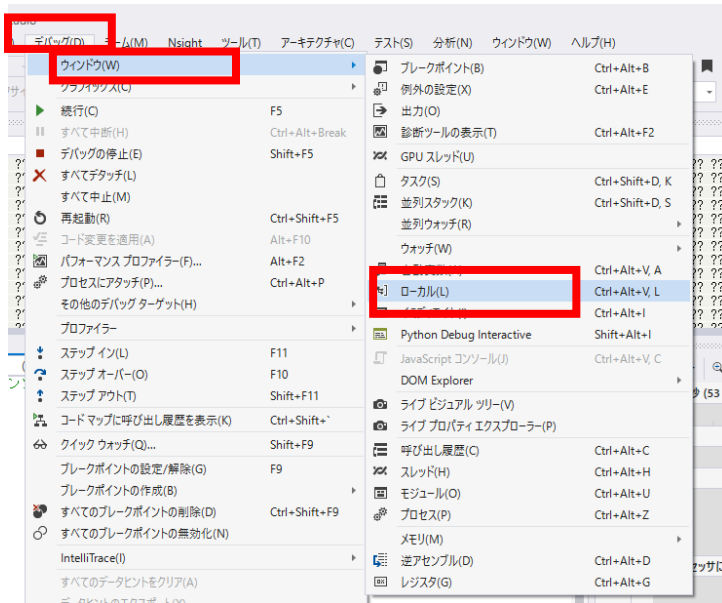
「デバッグ」
→ 「デバッグ開始」

- 「return a * 1.08;」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 double foo(double a)
7 {
8     [
9
10 ]
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f\n", foo(p));
16     p = 200;
17     printf("%f\n", foo(p));
18     return 0;
19 }
20
```

「return a * 1.08;」の
行で実行が
中断している

- 「return a * 1.08;」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。



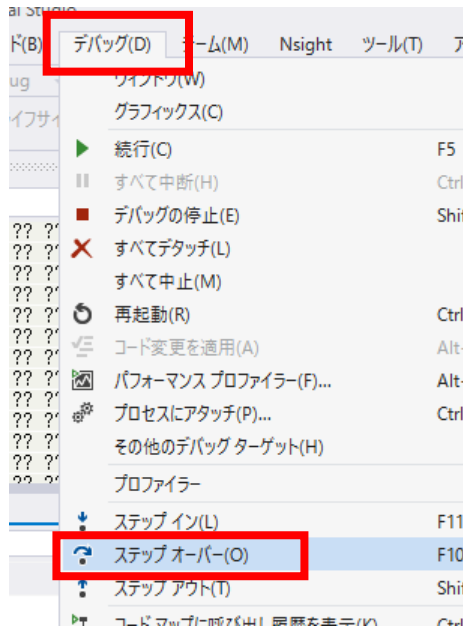
名前	値
a	120.00000000000000

② 変数名と値の対応表が表示される

変数 a の値 120

① 「デバッグ」
→ 「ウインドウ」
→ 「ローカル」

- ステップオーバーの操作を1回ずつ行いなさい。



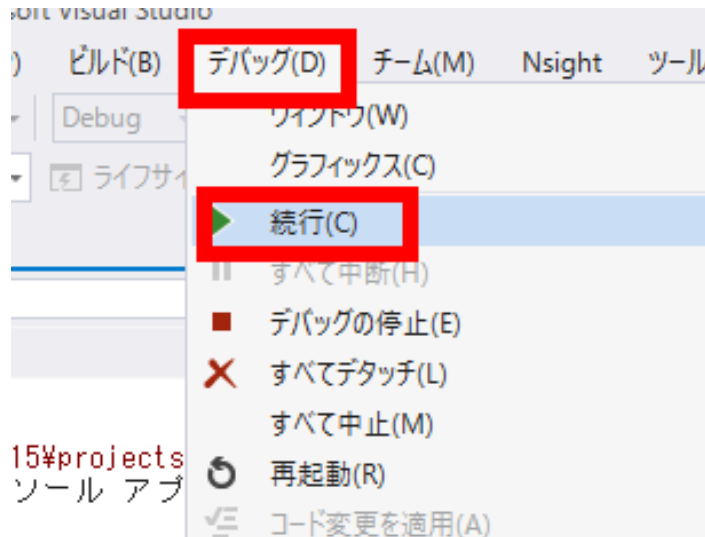
```
1 // ConsoleApplication10.cpp : :
2 //
3
4 #include "stdafx.h"
5
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f\n", foo(p));
16     p = 200;
17     printf("%f\n", foo(p));
18     return 0;
19 }
```

```
1 // ConsoleApplication10.cpp : :
2 //
3
4 #include "stdafx.h"
5
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f\n", foo(p));
16     p = 200;
17     printf("%f\n", foo(p));
18     return 0; ≤1ミリ秒経過
19 }
```

みどころ、
黄色の矢印が動く。
あるときは **foo 関数の中**。
あるときは **main 関数の中**

「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**」

サブルーチン呼び出しのための call 命令



```
1 // ConsoleApplication10.cpp : 二
2 //
3
4 #include "stdafx.h"
5
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f¥n", foo(p));
16     p = 200;
17     printf("%f¥n", foo(p));
18     return 0; // ≤1ミリ秒経過
19 }
20
```

```
p = 200;
0008183D movsd xmm0,mmword ptr [__real@40690000]
00081845 movsd mmword ptr [p],xmm0
printf("%f¥n", foo(p));
0008184A sub esp,8
0008184D movsd xmm0,mmword ptr [p]
00081852 movsd mmword ptr [esp],xmm0
00081857 call foo (0811B3h)
0008185C fstp qword ptr [esp]
0008185F push offset string "%f¥n" (086B30h)
00081864 call _printf (08131Bh)
00081869 add esp,0Ch
return 0;
```

アセンブリ言語の call 命令

「call foo . . .」

でサブルーチンの先頭へジャンプ

Visual Studio の C++ プログラム

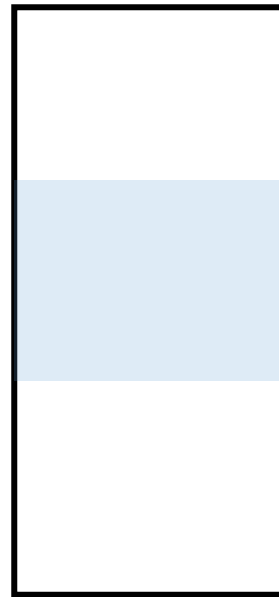
コンピュータでのスタックの作り方



メモリ

事前に、メモリ内に
スタック専用エリアを確保
(システムスタックエリア)

コンピュータでのスタックの作り方



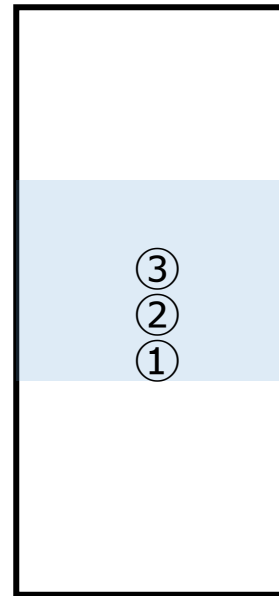
メモリ

最初は空

コンピュータでのスタックの作り方



①, ②, ③の順で
プッシュすると・・・



メモリ

一番最後にプッシュ
した③が一番上

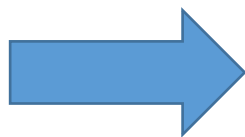
データが入る

13-2 サブルーチン呼び出しと 復帰

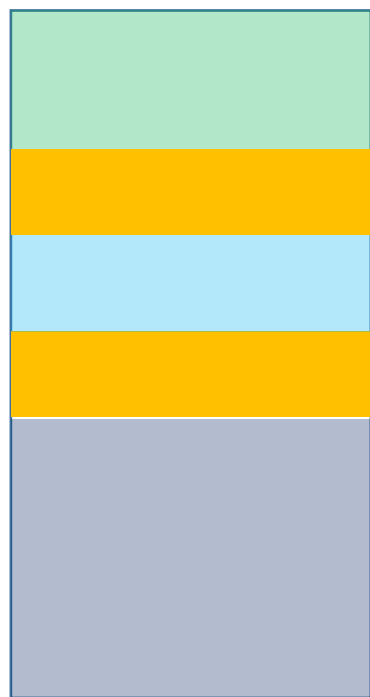
サブルーチンとは



サブルーチンを使うように、
プログラムを書き換え



類似の処理を1か所に
まとめたもの



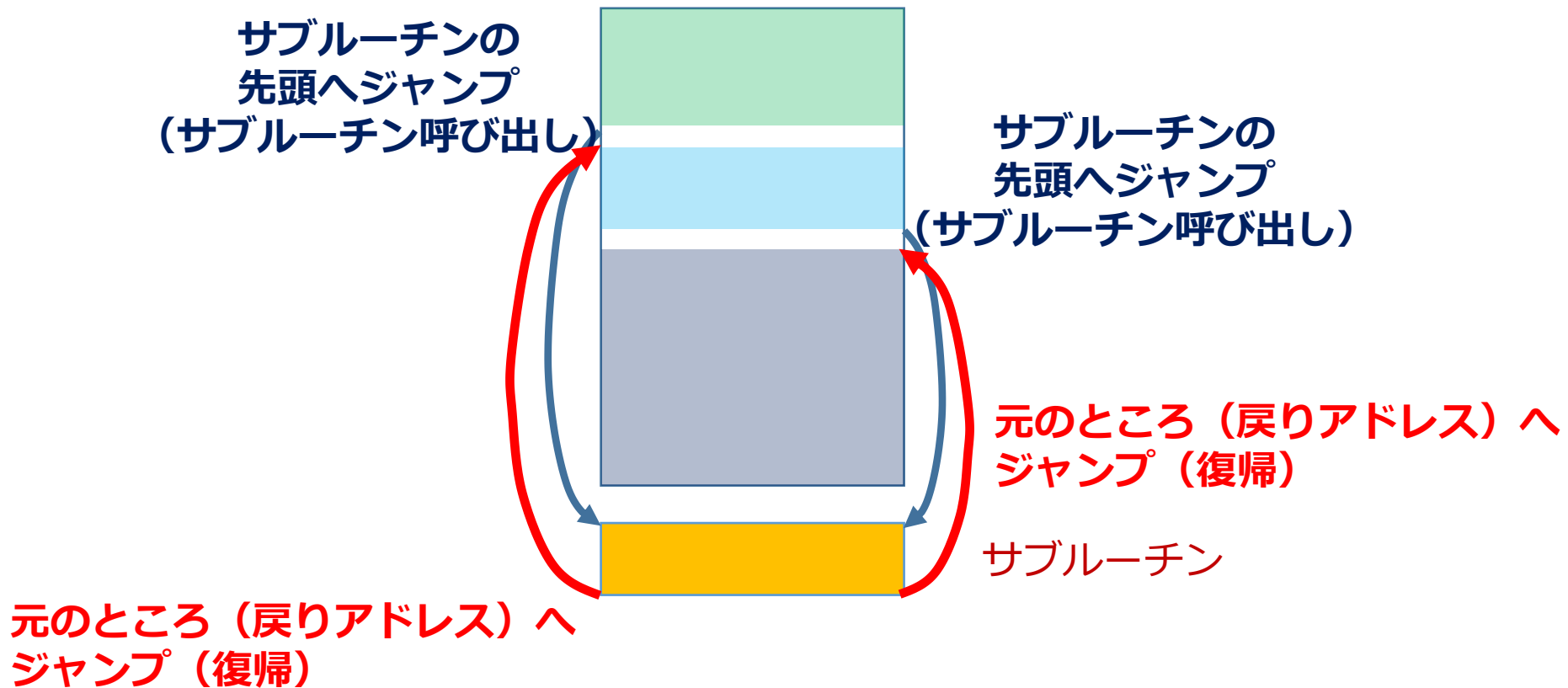
サブルーチンなし



サブルーチン

サブルーチンあり

サブルーチン呼び出しと復帰



サブルーチン呼び出しと復帰



- プログラムの中から,
- サブルーチンの先頭にジャンプ（サブルーチン呼び出し）して,
- あとで戻ってくる（復帰）

サブルーチン呼び出しと復帰



復帰

呼び出しのための
ジャンプ

復帰のための
ジャンプ

呼び出し

呼び出し

```
4      #include "stdafx.h"
5
6      double foo(double a)
7      {
8          return a * 1.08;
9      }
10
11     int main()
12     {
13
14         double p;
15         p = 120;
16         printf("%f¥n", foo(p));
17         p = 200;
18         printf("%f¥n", foo(p));
19         return 0;
20     }
```

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

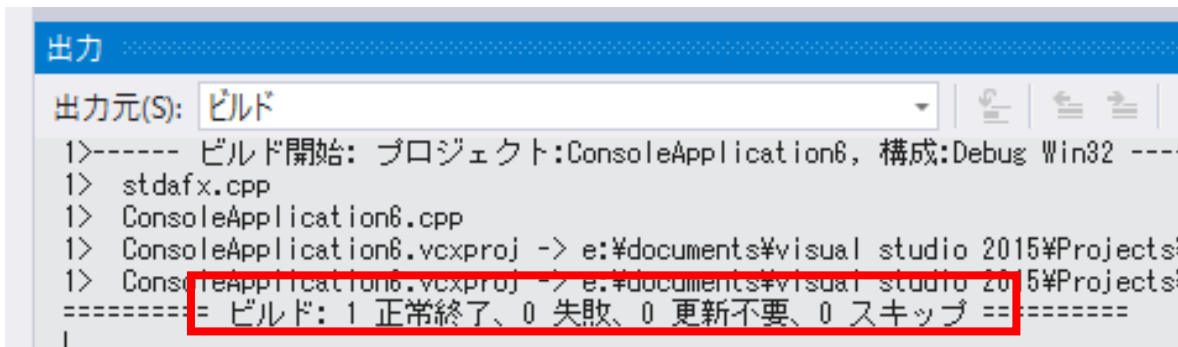
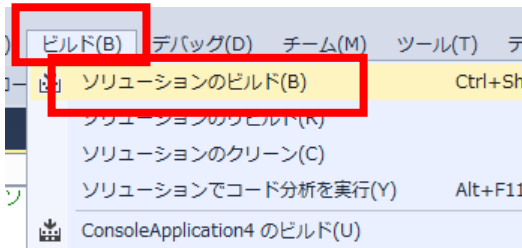
プロジェクトの「名前」は何でもよい

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

```
4      #include "stdafx.h"
5
6      double foo(double a)
7      {
8          return a * 1.08;
9      }
10
11     int main()
12     {
13         double p;
14         p = 120;
15         printf("%f¥n", foo(p));
16         p = 200;
17         printf("%f¥n", foo(p));
18         return 0;
19     }
20
```

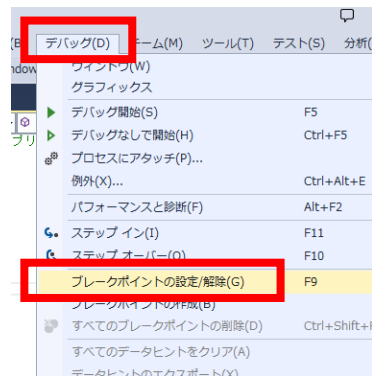
- ビルドしなさい。ビルドのあと「1 正常終了, 0 失敗」の表示を確認しなさい

→ 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 で「return a * 1.08;」の行に、ブレークポイントを設定しなさい

```
4 #include "stdafx.h"
5
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f¥n", foo(p));
16     p = 200;
17     printf("%f¥n", foo(p));
18     return 0;
19 }
```



```
4 #include "stdafx.h"
5
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f¥n", foo(p));
16     p = 200;
17     printf("%f¥n", foo(p));
18     return 0;
19 }
```

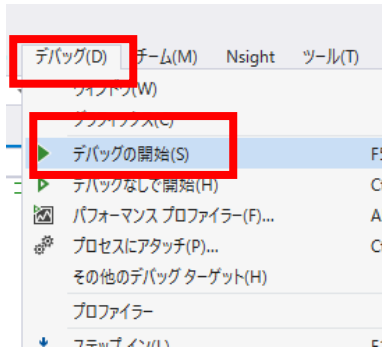


① 「return a * 1.08;」の行をマウスでクリック

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 で、デバッガーを起動しなさい。



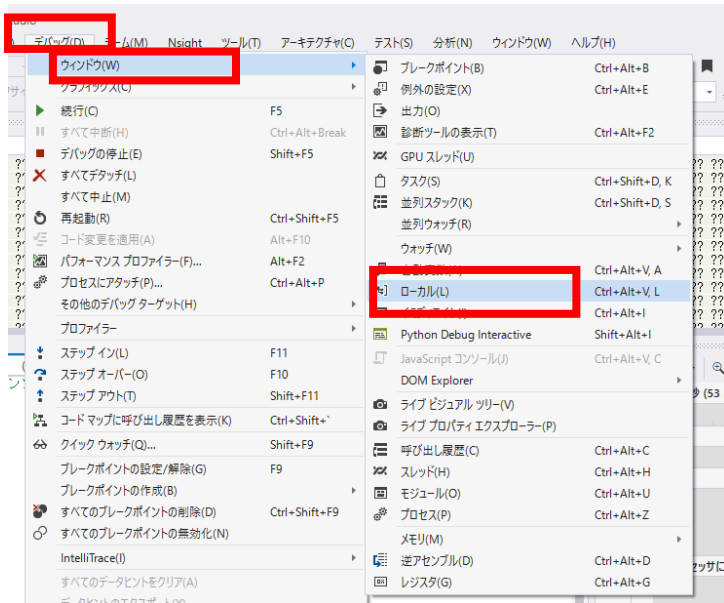
「デバッグ」
→ 「デバッグの開始」


- 「return a * 1.08;」 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 double foo(double a)
7 {
8     // ...
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f\n", foo(p));
16     p = 200;
17     printf("%f\n", foo(p));
18     return 0;
19 }
```

「return a * 1.08;」 の
行で実行が
中断している

- 「return a * 1.08;」 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。



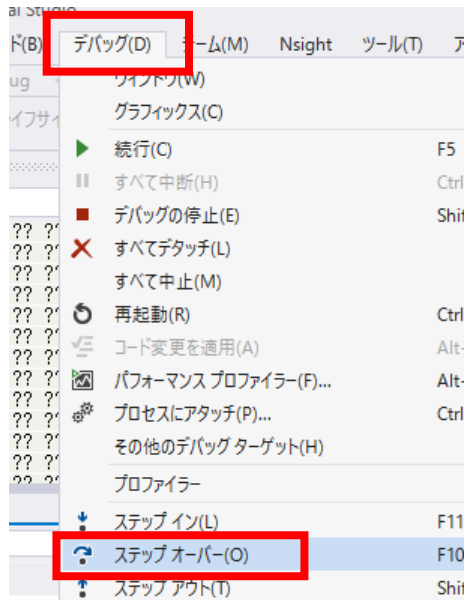
ローカル	
名前	値
 a	120.00000000000000

② 変数名と値の対応表が表示される

変数 **a** の値 **120**

① 「デバッグ」
→ 「ウィンドウ」
→ 「ローカル」

- ステップオーバーの操作を1回ずつ行いなさい。



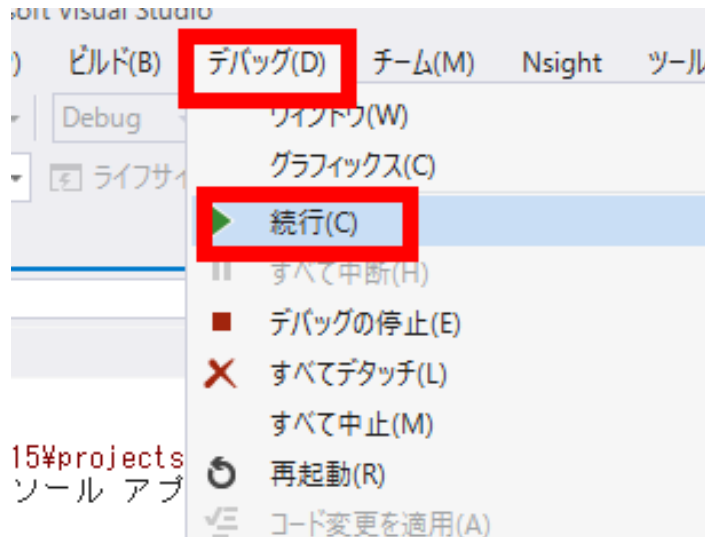
「デバッグ」
→ 「ステップオーバー」
(あるいは F10 キー)

```
1 // ConsoleApplication10.cpp : :
2 //
3
4 #include "stdafx.h"
5
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f\n", foo(p));
16     p = 200;
17     printf("%f\n", foo(p));
18     return 0;
19 }
```

```
1 // ConsoleApplication10.cpp : :
2 //
3
4 #include "stdafx.h"
5
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f\n", foo(p));
16     p = 200;
17     printf("%f\n", foo(p));
18     return 0; ≤1ミリ秒経過
19 }
```

みどころ、
黄色の矢印が動く。
あるときは **foo 関数の中**。
あるときは **main 関数の中**

- 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」

サブルーチン呼び出しのための call 命令



```
1 // ConsoleApplication10.cpp : 二
2 //
3
4 #include "stdafx.h"
5
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double p;
14     p = 120;
15     printf("%f¥n", foo(p));
16     p = 200;
17     printf("%f¥n", foo(p));
18     return 0; // ≤1ミリ秒経過
19 }
20
```

```
    p = 200;
0008183D movsd    xmm0,mmword ptr [__real@40690000]
00081845 movsd    mmword ptr [p],xmm0
    printf("%f¥n", foo(p));
0008184A sub     esp,8
0008184D movsd    xmm0,mmword ptr [p]
00081852 movsd    mmword ptr [esp],xmm0
00081857 call   foo (0811B3h)
0008185C fstp   qword ptr [esp]
0008185F push   offset string "%f¥n" (086B30h)
00081864 call   _printf (08131Bh)
00081869 add     esp,0Ch
    return 0;
```

アセンブリ言語の call 命令

「call foo . . .」

でサブルーチンの先頭へジャンプ

Visual Studio の C++ プログラム

- 次のように書き替えて，同じ手順を繰り返さない。

```
6 double foo(double a)
7 {
8     return a * 1.08;
9 }
10
11 int main()
12 {
13     double x[6] = { 30, 100, 50, 200, 80, 40 };
14     double y[6];
15     int i;
16     for (i = 0; i < 6; i++) {
17         y[i] = foo(x[i]);
18     }
19     return 0;
20 }
21
```

13-3 サブルーチン呼び出しに 関する演習

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

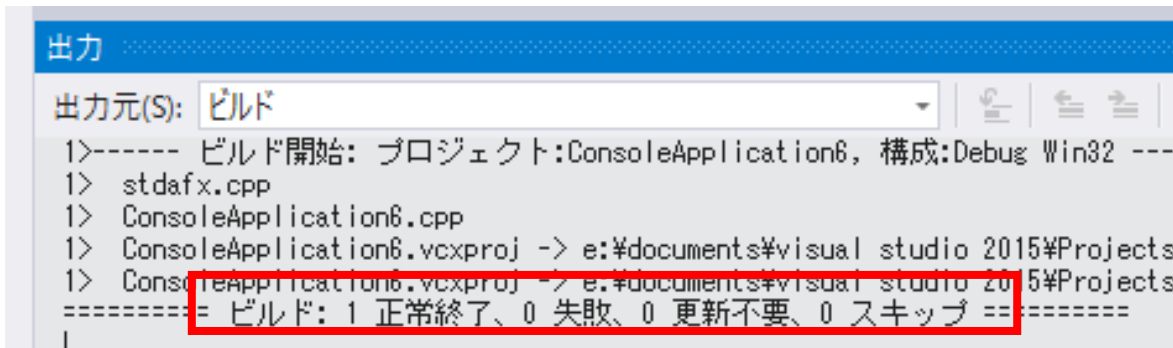
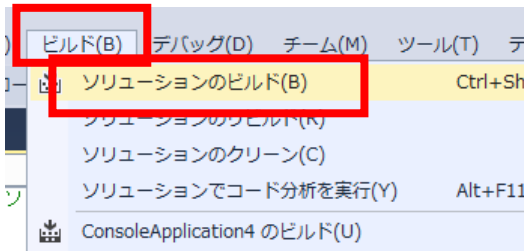
プロジェクトの「名前」は何でもよい

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

```
double foo(double a)
{
    return a * 1.08;
}
double bar(double x)
{
    return foo(x) * 100;
}
int main()
{
    double p;
    p = 12;
    printf("%f¥n", bar(p));
    p = 20;
    printf("%f¥n", bar(p));
    return 0;
}
```

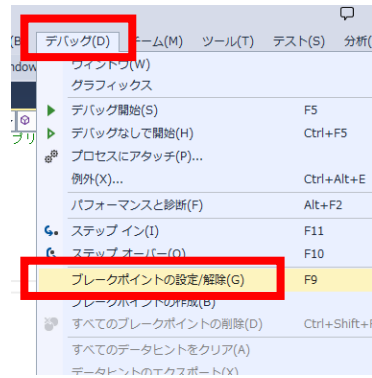
- ビルドしなさい。ビルドのあと「1 正常終了, 0 失敗」の表示を確認しなさい

→ 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 で「return a * 1.08;」の行に、ブレークポイントを設定しなさい

```
double foo(double a)
{
    return a * 1.08;
}
double bar(double x)
{
    return foo(x) * 100;
}
int main()
{
    double p;
    p = 12;
    printf("%f¥n", bar(p));
    p = 20;
    printf("%f¥n", bar(p));
    return 0;
}
```



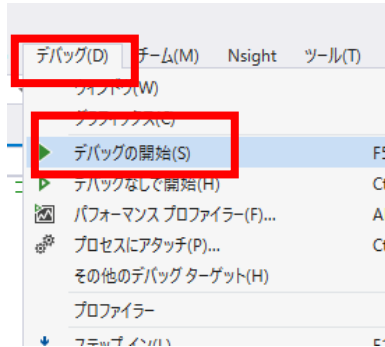
```
6
7
8 return a * 1.08;
9
10 double bar(double x)
11 {
12     return foo(x) * 100;
13 }
14 int main()
15 {
16     double p;
17     p = 12;
18     printf("%f¥n", bar(p));
19     p = 20;
20     printf("%f¥n", bar(p));
21     return 0;
22 }
??
```

① 「return a * 1.08;」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」

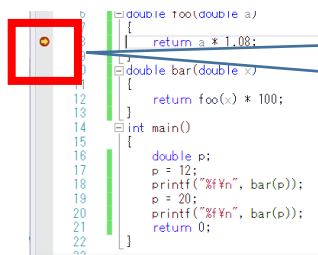
③ ブレークポイントが設定されるので確認。赤丸がブレークポイントの印

- Visual Studio で、デバッガーを起動しなさい。



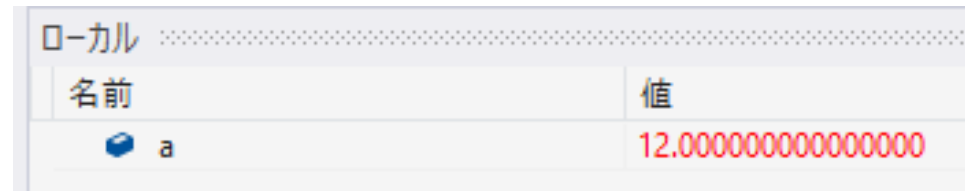
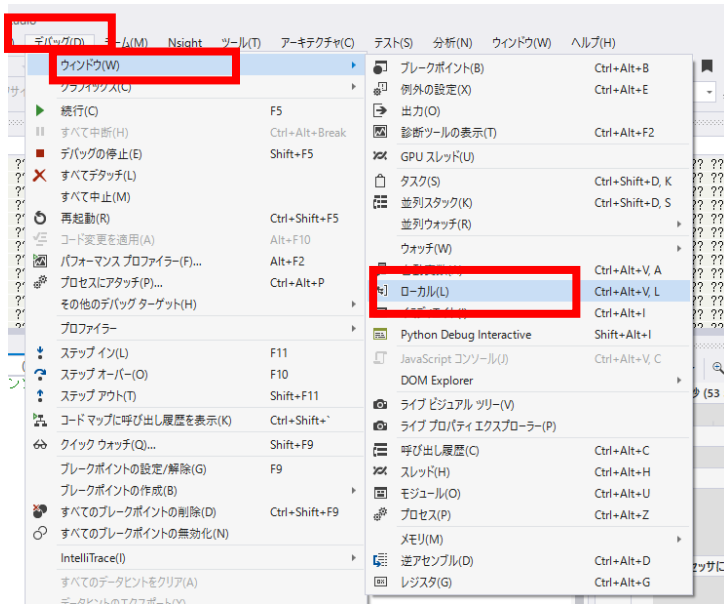
「デバッグ」
→ 「デバッグの開始」

- 「return a * 1.08;」 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



「return a * 1.08;」 の
行で実行が
中断している

- 「return a * 1.08;」 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 a screenshot of the 'ローカル' (Locals) window in the IDE. The window has a header 'ローカル' and a table with two columns: '名前' (Name) and '値' (Value). The table contains one entry: 'a' with a value of '12.0000000000000000'. The value is displayed in red text.

名前	値
a	12.0000000000000000

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 a の値 12

```

6  double foo(double a)
7  {
8      return a * 1.08;
9  }
10 double bar(double x)
11 {
12     return foo(x) * 100;
13 }
14 int main()
15 {
16     double p;
17     p = 12;
18     printf("%f¥n", bar(p));
19     p = 20;
20     printf("%f¥n", bar(p));
21     return 0;
22 }

```

```

6  double foo(double a)
7  {
8      return a * 1.08;
9  }
10 double bar(double x)
11 {
12     return foo(x) * 100; ≤1ミリ秒経過
13 }
14 int main()
15 {
16     double p;
17     p = 12;
18     printf("%f¥n", bar(p));
19     p = 20;
20     printf("%f¥n", bar(p));
21     return 0;
22 }

```

ローカル

名前	値
a	12.0000000000000000

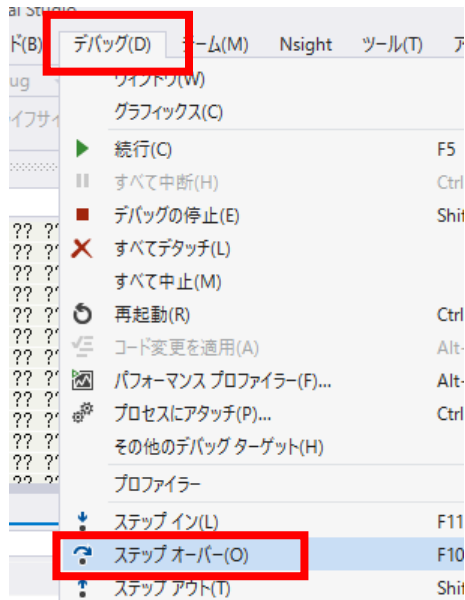
関数 **foo** の中では、**変数 a** だけが使える
(変数 x, p は使えない)

ローカル

名前	値
foo が返されました	12.9600000000000001
x	12.0000000000000000

関数 **bar** の中では、**変数 x** だけが使える
(変数 a, p は使えない)

- ステップオーバーの操作を1回ずつ行いなさい。



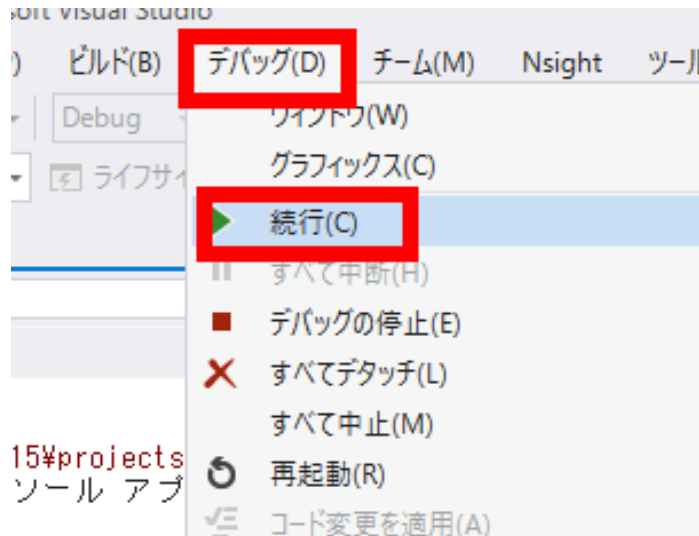
「デバッグ」
→ 「ステップオーバー」
(あるいは F10 キー)

```
6  double foo(double a)
7  {
8      return a * 1.08;
9  }
10 double bar(double x)
11 {
12     return foo(x) * 100;
13 }
14 int main()
15 {
16     double p;
17     p = 12;
18     printf("%f\n", bar(p));
19     p = 20;
20     printf("%f\n", bar(p));
21     return 0;
22 }
```

```
6  double foo(double a)
7  {
8      return a * 1.08;
9  }
10 double bar(double x)
11 {
12     return foo(x) * 100; ≤1ミリ秒経過
13 }
14 int main()
15 {
16     double p;
17     p = 12;
18     printf("%f\n", bar(p));
19     p = 20;
20     printf("%f\n", bar(p));
21     return 0;
22 }
```

みどころ、
黄色の矢印が動く。
あるときは **foo 関数の中**。
あるときは **bar 関数の中**。
あるときは **main 関数の中**。

- 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」

関数呼び出しのための call 命令



```
double foo(double a)
{
    return a * 1.08;
}
double bar(double x)
{
    return foo(x) * 100;
}
int main()
{
    double p;
    p = 12;
    printf("%f¥n", bar(p));
    p = 20;
    printf("%f¥n", bar(p));
    return 0;
}
```

```
009C16F9 push    ebx
009C16FA push    esi
009C16FB push    edi
009C16FC lea    edi,[ebp-0D0h]
009C1702 mov    ecx,34h
009C1707 mov    eax,0CCCCCCCCh
009C170C rep stos dword ptr es:[edi]
        return foo(x) * 100;
009C170E sub    esp,8
009C1711 movsd  xmm0,mmword ptr [x]
009C1716 movsd  mmword ptr [esp],xmm0
009C171B call   foo (09C11B3h)
009C1720 add    esp,8
009C1723 fstp   qword ptr [ebp-0C8h]
009C1729 movsd  xmm0,mmword ptr [ebp-0C8h]
009C1731 mulsd  xmm0,mmword ptr [__real@4059000000000000 (09C8B68h)]
009C1739 movsd  mmword ptr [ebp-0D0h],xmm0
009C1741 fld   qword ptr [ebp-0D0h]
}
009C1747 pop    edi
009C1748 pop    esi
009C1749 pop    ebx
009C174A add    esp,0D0h
```

アセンブリ言語の call 命令
「call foo . . .」

でサブルーチンの先頭へジャンプ

Visual Studio の C++ プログラム