

2021 計算機プログラミング演習

第12回
ユーザ定義関数

先週の解説 (おみくじプログラム)

プログラム作成の要件

- おみくじは1回の試行で1つのくじがでること
- 引いたくじの内容はコマンドプロンプト上に表示させる
- くじの確率は、**大吉が1/10, 中吉が2/10, 吉3/10, 末吉が3/10, 凶が1/10**とする
- 本プログラムは擬似乱数を用いる
 - ✓ 擬似乱数生成にはrand()関数を用いる
 - ✓ 必要なヘッダーファイルについては自分で調べる
 - ✓ rand()関数の記述の前には, srand(time(NLL));を記述すること
 - ✓ rand()を用いてランダムに0から9の整数を生成する

実行結果（おみくじプログラム）

```
コマンドプロンプト
Microsoft Windows [Version 10.0.19042.1055]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nakano>cd C:\Users\Nakano\source\repos\omikuj\Debug

C:\Users\Nakano\source\repos\omikuj\Debug>omikuj
おみくじプログラム開始する
末吉

C:\Users\Nakano\source\repos\omikuj\Debug>omikuj
おみくじプログラム開始する
末吉

C:\Users\Nakano\source\repos\omikuj\Debug>omikuj
おみくじプログラム開始する
大吉

C:\Users\Nakano\source\repos\omikuj\Debug>omikuj
おみくじプログラム開始する
吉

C:\Users\Nakano\source\repos\omikuj\Debug>omikuj
おみくじプログラム開始する
末吉

C:\Users\Nakano\source\repos\omikuj\Debug>omikuj
おみくじプログラム開始する
大吉

C:\Users\Nakano\source\repos\omikuj\Debug>omikuj
おみくじプログラム開始する
凶

C:\Users\Nakano\source\repos\omikuj\Debug>omikuj
おみくじプログラム開始する
吉
```

何回か試行してみて、設定している確率でおみくじが出ているか確認すること！

大吉の確率1/10

中吉の確率2/10

吉の確率が3/10

末吉の確率3/10

凶の確率1/10

プログラム例 おみくじプログラム

```
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <time.h>
12
13 int main(void) {
14     printf("おみくじプログラム開始する\n");
15     srand(time(NULL));
16     int num = rand() % 10;
17
18     if (num == 0)
19     {
20         printf("大吉\n");
21     }
22
23     if (num == 1 || num == 2)
24     {
25         printf("中吉\n");
26     }
27
28     if (num == 3 || num == 4 || num == 5)
29     {
30         printf("吉\n");
31     }
32
33     if (num == 6 || num == 7 || num == 8)
34     {
35         printf("末吉\n");
36     }
37
38     if (num == 9)
39     {
40         printf("凶\n");
41     }
42
43     return 0;
44 }
```

rand()関数の出力を10で割り、余りを求める
ことで0から9の整数の乱数を生成する

あとは、設定した確率に応じて整数を
振り分けるだけ

関数とは

これまで演習や課題でプログラムを作成したが、例えばこれらのプログラムを別のプログラム(main関数)でも利用したい場合を考えよう

- 必要な変数や処理部分をコピー＆ペーストをする

確かにこれで可能ではあるが、使いたいプログラムの数が多ければ多いほど、main関数の記述が膨大になり、煩雑なプログラムになる。

そこで、以前作成したプログラムの再利用を目的として、プログラムを再利用しやすい形に部品化を行う

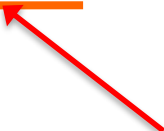
プログラムでは、この部品のことを関数と呼ぶ

ユーザー定義関数

- 実は関数の記述は既に行っている.
- main()もmain関数と呼ばれる関数である
- 記述方法はユーザー定義関数もmain関数と同様である

```
#include <stdio.h>
```

```
int main(void)
{
    return 0;
}
```



引数(引数の詳細に関しては後述する)

ユーザー定義関数

これまではmain関数内にプログラムでやりたい処理を記述していた

```
#include <stdio.h>
```

```
int main(void)  
{
```

ここに記述

```
    return 0;  
}
```

ユーザー定義関数

以下の様な消費税計算を行う関数を独立して作成する.

```
int tax (int kakaku)
{
    int zeikomi = 0; //初期化

    zeikomi = 1.08 * kakaku; //計算

    printf ( "%d¥n", zeikomi ); //表示

    return 0;
}
```


自作関数の記述位置

1. main関数の後に記述

```
#include <stdio.h>
```

```
int main(void)
{
    return 0;
}
```

```
int tax(int kakaku)
{
    int zeikomi =0; //初期化
    zeikomi = 1.08*kakaku; //計算
    printf("%d¥n", zeikomi); //表示
    return 0;
}
```

2. main関数の前に記述

```
int tax(int kakaku)
{
    int zeikomi =0; //初期化
    zeikomi = 1.08*kakaku; //計算
    printf("%d¥n", zeikomi); //表示
    return 0;
}
```

```
#include <stdio.h>
```

```
int main(void)
{
    return 0;
}
```

記述の位置としてはどちらでもよいが、ただし1.の場合はこのままでは関数を使うことはできない

ユーザー定義関数の使用

1. main関数の後に記述

```
#include <stdio.>
```

```
int tax(int); //プロトタイプ宣言
```

```
int main(void)
{
    return 0;
}
```

```
int tax(int kakaku)
{
    int zeikomi = 0; //初期化
    zeikomi = 1.08*kakaku; //計算
    printf("%d¥n", zeikomi); //表示
    return 0;
}
```

- 1.の場合main関数の前に作成した自作関数の簡単なリストを記述する必要がある
- このリスト記述を
プロトタイプ宣言と呼ぶ
- プロトタイプ宣言により関数の位置を気にする必要がなくなるが、基本的にはmain関数の後ろに記述する方が見やすく良い

ユーザー定義関数の使用

```
#include <stdio.>
```

```
int tax(int);    //プロトタイプ宣言
```

```
int main(void)
```

```
{  
    tax (100);  
    return 0;  
}
```

実引数

仮引数

```
int tax(int kakaku)  
{  
    int zeikomi =0; //初期化  
    zeikomi = 1.08*kakaku; //計算  
    printf("%d¥n", zeikomi); //表示  
    return 0;  
}
```

- 自作関数を使用する場合はmain関数内で自作関数を呼び出す必要がある

- 関数に任意の値を渡す仕組みを引数という

実引数

main関数内で関数の呼び出しと同時に渡す値のことを指す

仮引数

自作関数で定義された変数の型と変数名のことを指す

プログラム例1

ソースコード

実行

```
1  #include <stdio.h>
2
3  int tax(int);
4
5  int main(void) {
6      int value = 0;
7      value = tax(100);
8      return 0;
9  }
10
11 int tax(int kakaku)
12 {
13     int zeikomi = 0;
14     zeikomi = 1.08 * kakaku;
15     printf("%d\n", zeikomi);
16     return 0;
17 }
18
```

```
C:\Users\Nakano\source\repos\tax\Debug>tax
108
```

戻り値

引数を用いて関数に値を渡すが, ここでは関数から値を返す記述について解説する(実はもうその記述はしている)

```
1  #include <stdio.h>
2
3  int tax(int); ← プロトタイプ宣言
4
5  int main(void) {
6      int value = 0;
7      value = tax(100);
8      return 0;
9  }
10
11  int tax(int kakaku)
12  {
13      int zeikomi = 0;
14      zeikomi = 1.08 * kakaku;
15      printf("%d\n", zeikomi);
16      return 0; ← 関数が戻り値(戻り値)
17  }              の型を示している
18
```

関数が戻り値を返すことを示している
ただし, zeikomiを返す場合は"return zeikomi;"と記述

戻り値を返さない場合は, 仮引数を"void"と記述する

戻り値を用いたプログラム例2

プログラム例1との違いをきちんと理解すること！！

ソースコード

実行

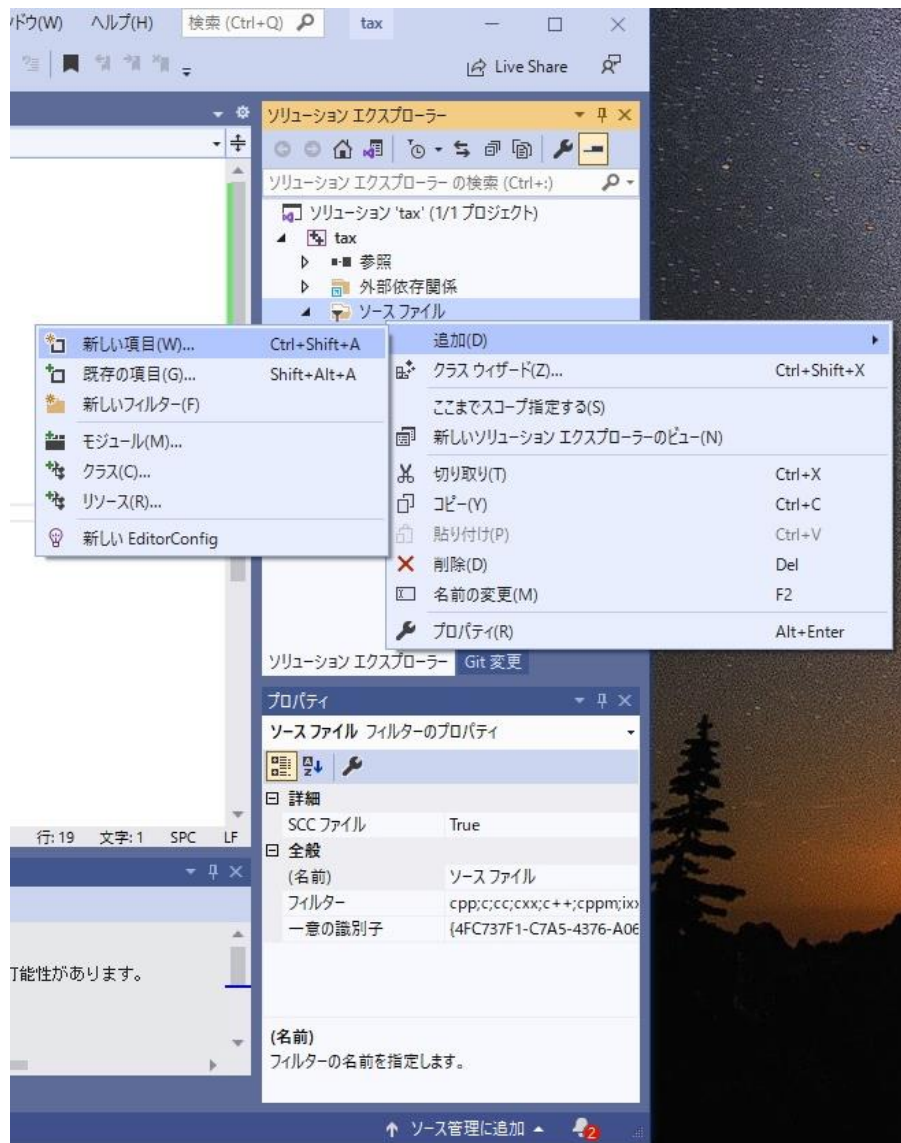
```
1  #include <stdio.h>
2
3  int tax(int);
4
5  int main(void) {
6      int value = 0;
7      value = tax(100);
8      printf("%d\n", value);
9      return 0;
10 }
11
12 int tax(int kakaku)
13 {
14     int zeikomi = 0;
15     zeikomi = 1.08 * kakaku;
16     return zeikomi;
17 }
18
```

```
C:\Users\Nakano\source\repos\tax\Debug>tax
108
```

ユーザ定義関数を 別ファイルに独立させよう！

- プロトタイプ宣言を行うと、ユーザ定義関数の配置 (`main()` の前、後) のみならず、同一ファイル内に記述しなくても、ユーザ定義関数を探すようになる
- 作成したユーザ定義関数が汎用性があるなら、他のプログラムからでも使用できるようにしたい
 - 過去に作ったユーザ定義関数が利用できる
 - 実績のある (バグがない) ユーザ定義関数

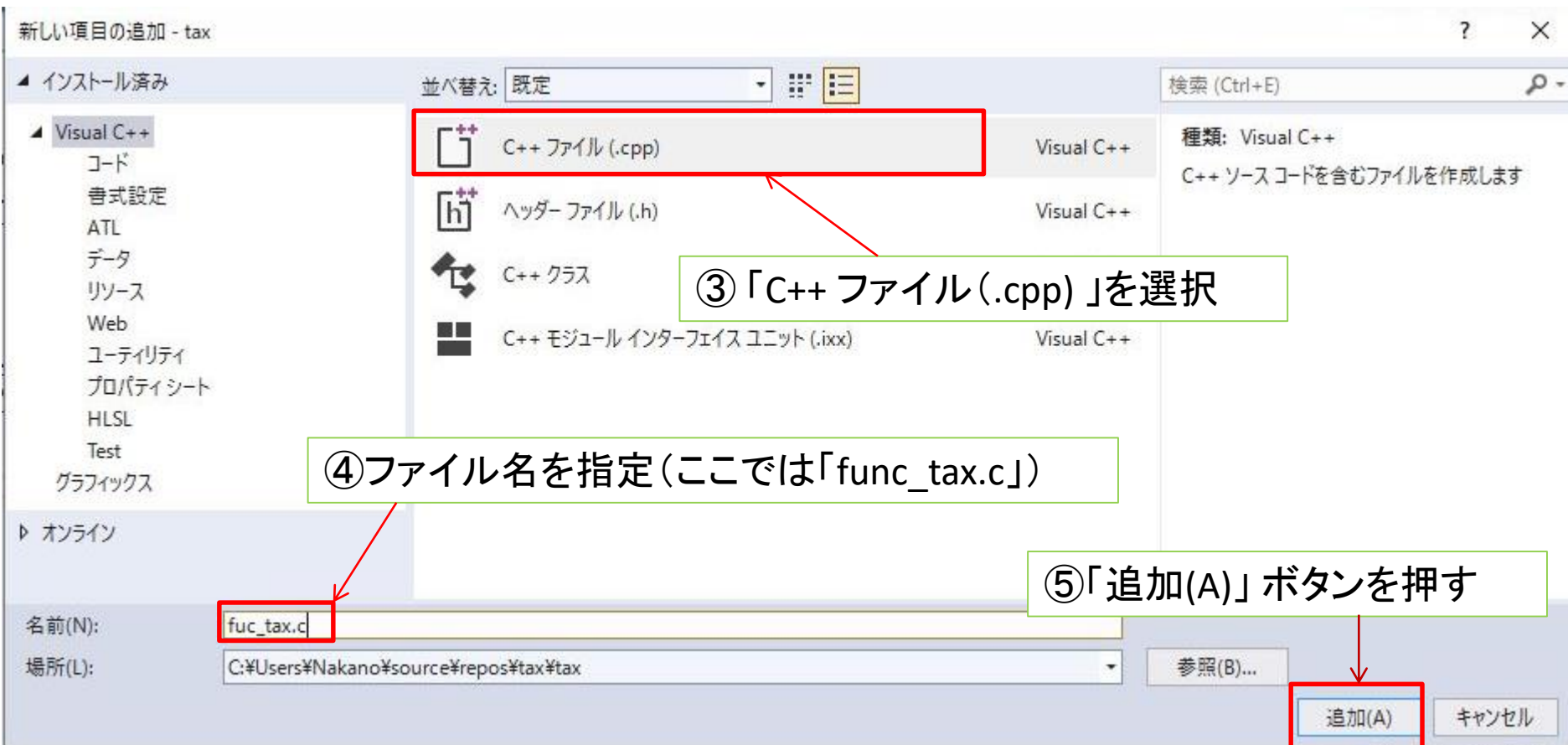
ユーザ定義関数を別ファイルにする(1)



①画面右の「ソリューションエクスプローラ」から「ソースファイル」を右クリック

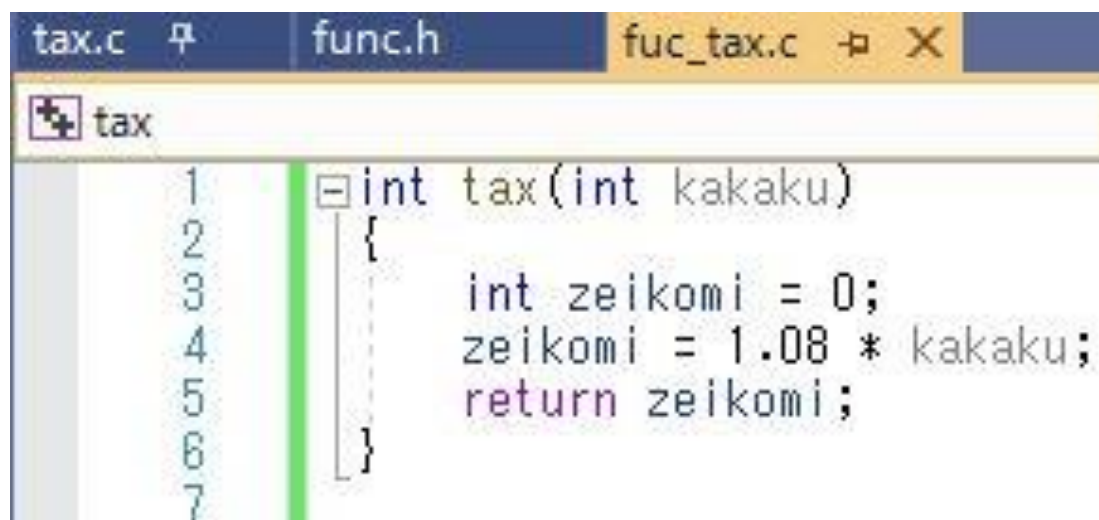
②「追加」->「新しい項目」を選択

ソースファイルの追加



ユーザー定義関数のファイル作成

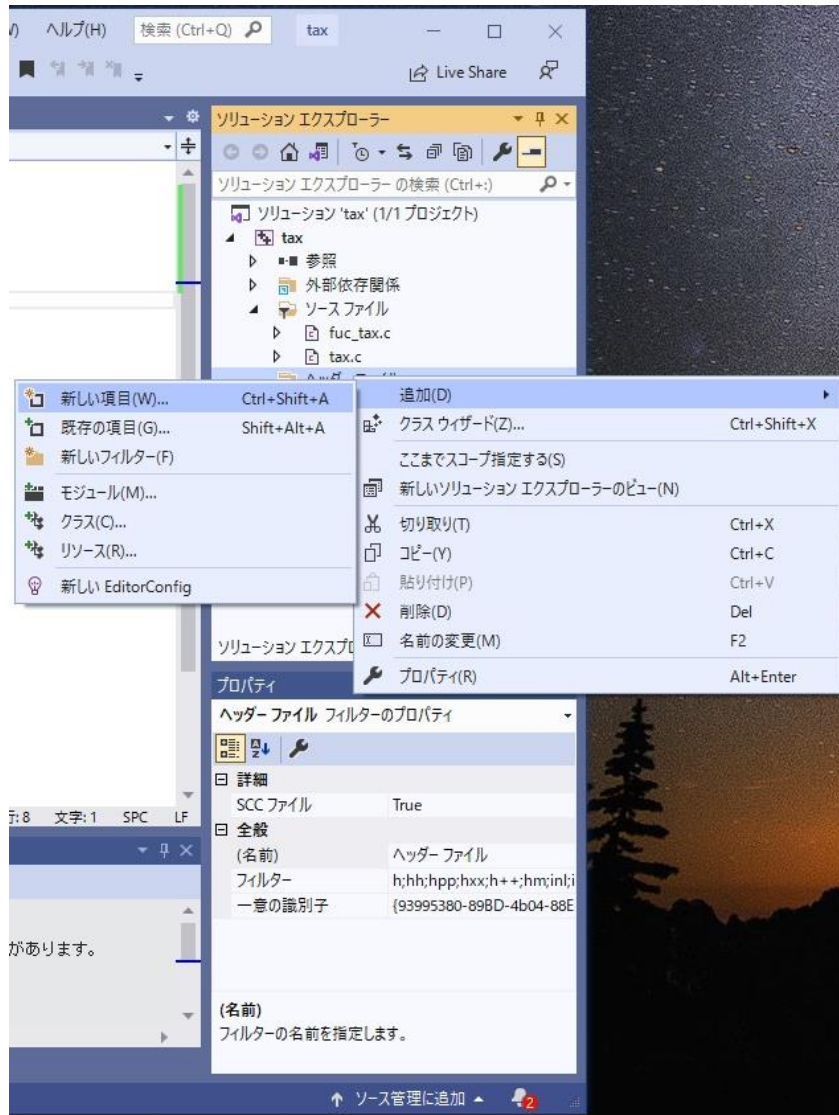
追加したソースファイル func_tax.c に自作関数を記述



The screenshot shows a code editor with three tabs: tax.c, func.h, and fuc_tax.c. The fuc_tax.c tab is active, displaying the following C code:

```
1 int tax(int kakaku)
2 {
3     int zeikomi = 0;
4     zeikomi = 1.08 * kakaku;
5     return zeikomi;
6 }
7
```

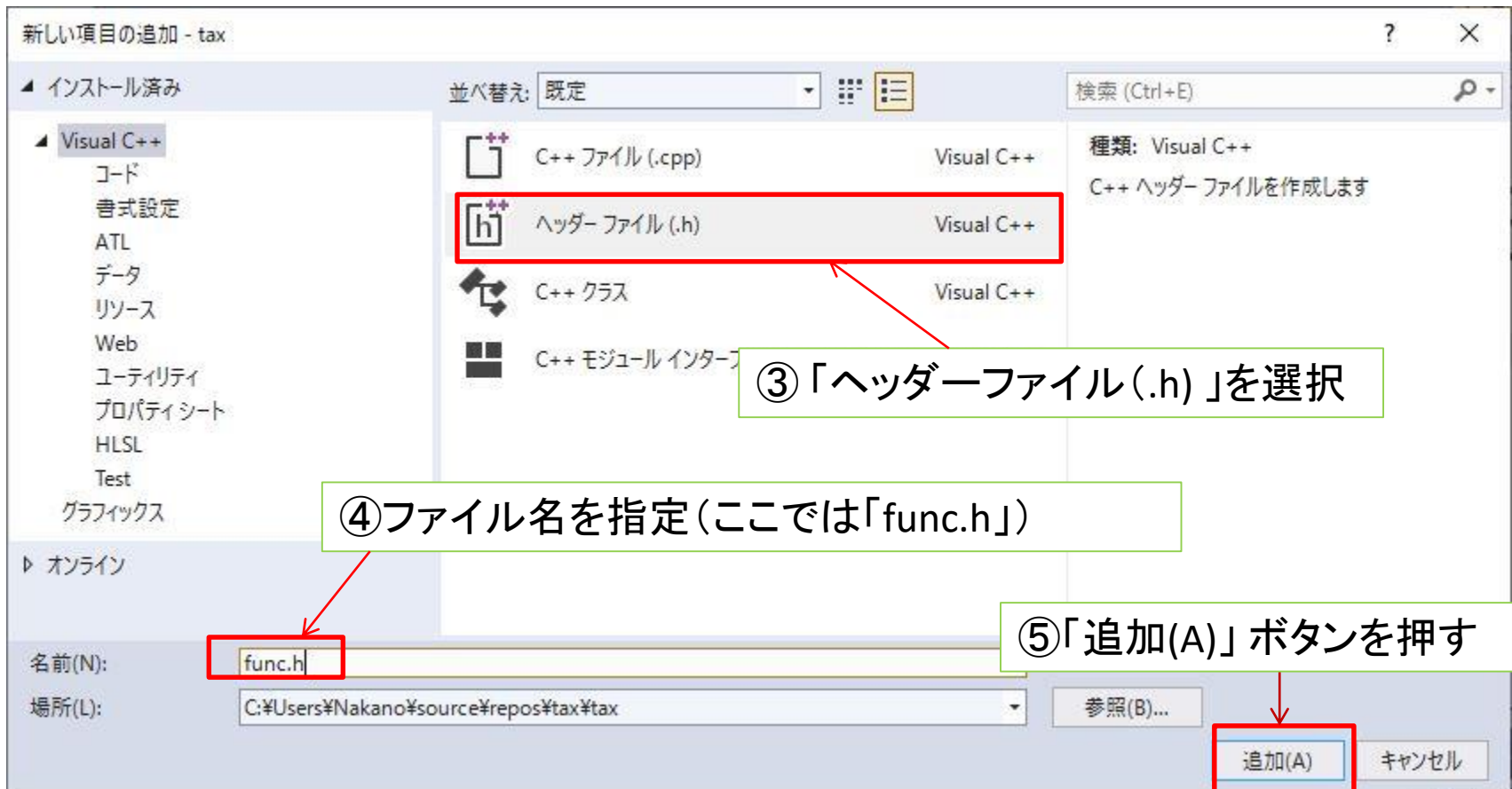
ヘッダーファイルの追加



①画面右の「ソリューションエクスプローラ」から「ヘッダーファイル」を右クリック

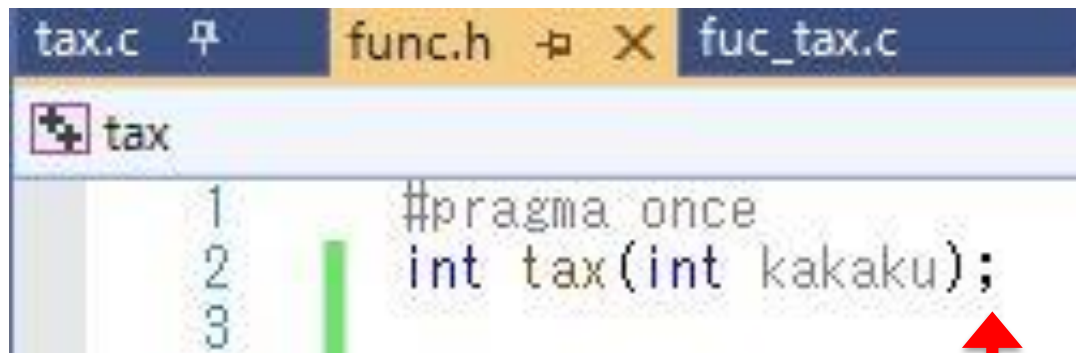
②「追加」->「新しい項目」を選択

ヘッダーファイルの追加



ヘッダーファイルの追加

追加したヘッダーファイル func.h
の1行目を記述(セミicolon ; 付けること)



The screenshot shows a code editor with three tabs at the top: 'tax.c', 'func.h', and 'fuc_tax.c'. The 'tax.c' tab is active, showing a file named 'tax'. The code in the editor is as follows:

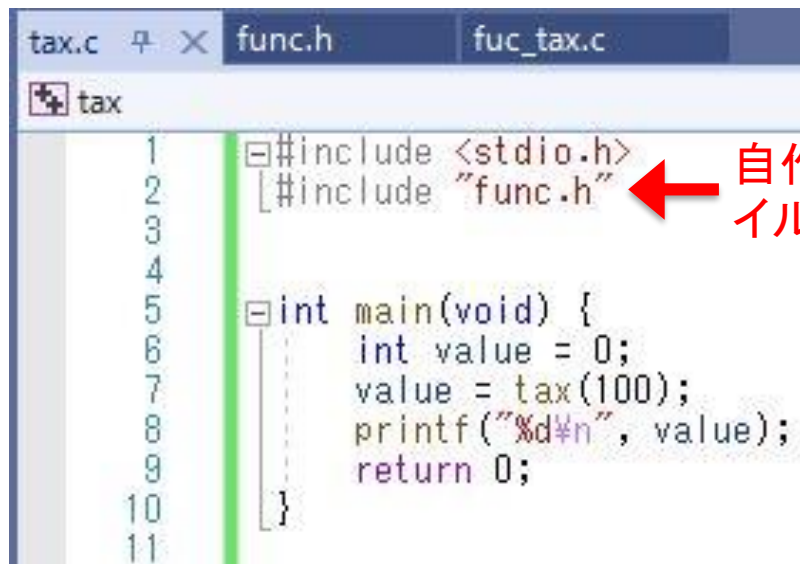
```
1 #pragma once
2 int tax(int kakaku);
3
```

A red arrow points to the semicolon at the end of line 2, highlighting the instruction to not forget it.

セミicolon 忘れないこと

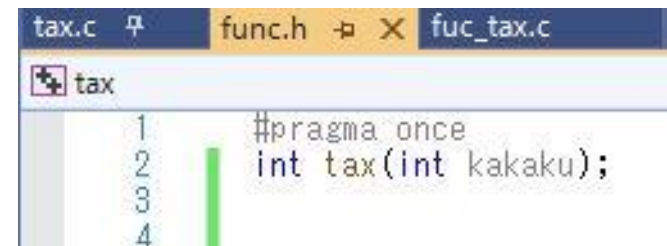
ファイルの確認, ビルド, 実行

1. main()関数が記述されている tax.c に自作のヘッダーファイル(func.h)をインクルード
2. 3つのファイル(tax.c, func_tax.c, func.h)があることを確認した後にビルド, 実行

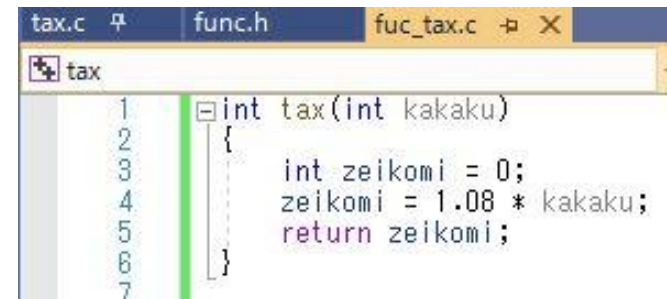


```
tax.c  7  X  func.h  fuc_tax.c
tax
1  #include <stdio.h>
2  #include "func.h"
3
4
5  int main(void) {
6      int value = 0;
7      value = tax(100);
8      printf("%d\n", value);
9      return 0;
10 }
11
```

自作のヘッダーファイル
をインクルード



```
tax.c  7  X  func.h  fuc_tax.c
tax
1  #pragma once
2  int tax(int kakaku);
3
4
```



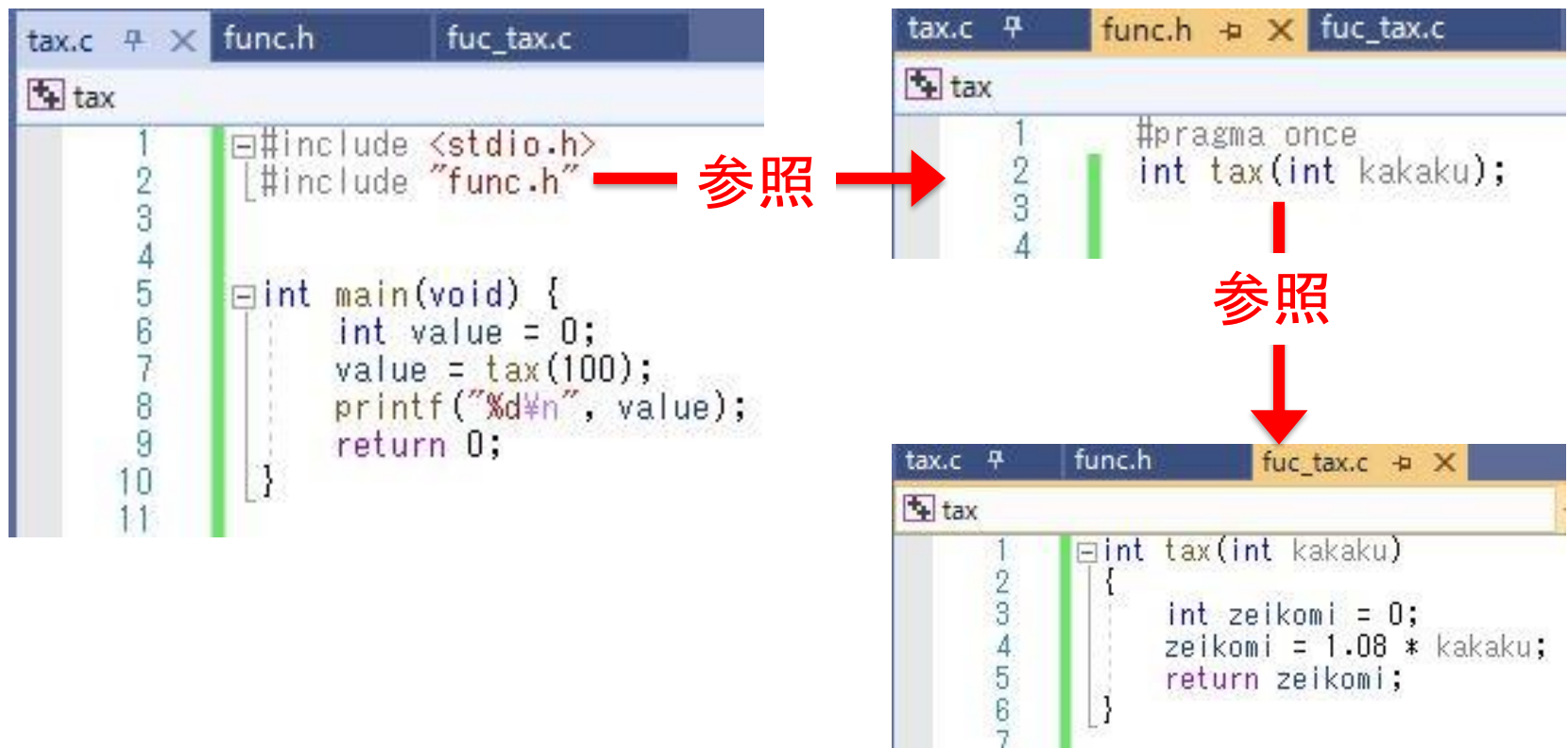
```
tax.c  7  func.h  fuc_tax.c  X
tax
1
2  int tax(int kakaku)
3  {
4      int zeikomi = 0;
5      zeikomi = 1.08 * kakaku;
6      return zeikomi;
7  }
```

tax.c には main()関数 のみを記述
tax.c にプロトタイプ宣言の記述は不要

```
C:\Users\Nakano\source\repos\tax\Debug>tax
108
```

各ファイルの関係性

1. main()関数に記述されてソースにインクルードされているヘッダーファイルを参照
2. ヘッダーファイル内に定義されている関数を参照



分割コンパイルとは？（１）

- 複数のソフトウェア部品を統合してプログラムを作成
 - 複数のプログラマがひとつのプログラムを共同開発する際、単一のファイルを対象に作業するのは効率が悪い。
 - プログラムを小さく分割し、部品として取り扱う。
- ソフトウェアの再利用性
 - 以前に作った部品を再利用できる
 - 何度も同じバグに悩む必要はない

分割コンパイルとは？（２）

- プログラムを再利用しやすいように複数のソースファイル (*.c, *.h) に分割
- それぞれをコンパイルして、オブジェクトファイルを作成
- 最後にリンカで統合し、実行形式ファイルを作成
- 欠点
 - ファイル間の関係がわかりにくくなる。
 - ドキュメント(プログラムについての説明)の必要性
 - コンパイルの段階的な動作を理解する必要がある。(プリプロセッサ、コンパイラ、リンカー)