

# 2021 計算機プログラミング演習

## 第3回

printf() と変数( int, float, double)

# プログラムの書き方(1)

- プログラムは上から下に向かって順番に書いていく
  - 実行の順番が変わる命令がいくつかあるが、上から下へ実行が進むのが原則である
- C 言語では「関数」を組み合わせてプログラムを作る
  - 「関数」(function) ?
    - function 【名詞】 関数、機能
    - 一般に C 言語では「関数」と呼ばれる
      - ひとまとまりの小さなプログラムのこと
      - 「機能」と言った方が初心者には分かりやすいかもしれない
  - 複数の命令をまとめて動作をさせるブロックを作成する
    - 目的の機能を果たす小さなプログラムを一つのブロックにまとめる。(関数)
    - いちど関数を作ったら、何度でも利用できる

# プログラムの書き方(2)

- 関数(function)とは？
  - printf()などの()が付く命令
  - 変数と区別するために明示的に()を付ける
  - ユーザーが作成することが可能
  - (関数内部で)複数の命令で構成し, まとまった機能を提供する
- C言語では `main()` 関数から実行が始まる
  - プログラムに1つの `main()` 関数が必要
    - `main()` 関数がプログラム中ないとビルドできない
    - `main()` 関数が複数あってもダメ, 一つだけ
  - `main()` 関数より前に何らかのプログラムが書かれていることはあるが, 実質的な実行は `main()` 関数から始まる

# プログラムの書き方(3)

- インデント(字下げ) プログラムの構造(ブロック)を示す
  - **TAB キー**を使って字下げを行う
    - Visual Studio では**自動的に**インデントが行われる
    - スペースキーでインデントを行うのではない
      - 字下げの幅がまちまちになり、プログラムの構造が分かりにくくなる

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

```
int main(void)
```

```
{
```

**TAB**

```
int ii, jj;  
int ans ;
```

```
for (ii=1;ii<=9;ii++) {
```

**TAB**

```
for (jj=1;jj<=9;jj++) {
```

**TAB**

```
ans = ii * jj ;  
printf("%d * %d = %d¥n",ii,ans);
```

```
}
```

```
}
```

```
return 0 ;
```

```
}
```

{ } が多重化したときなどにプログラムの構造を明確化するために字下げを行っている

例に示すプログラムは現時点では理解する必要はない

# エラーやバグについて

- プログラムの作成にエラーやバグはつきもの
  - エラー(error): プログラムを作成する上での誤り(入力ミスなど)
  - バグ(bug): プログラムを実行する上での不具合や, プログラム作成者が意図しない結果. 解決が難しいことがある
- エラーやバグは出るのが「**あたりまえ**」
  - 出たからと言ってがっかりする必要はない
  - ただし、それを克服することが必要
    - プログラムの文法についての正しいルールを知ること
    - **論理的な考え方を身につける**
- **エラーメッセージを読もう！**
  - エラー内容を知ることは実は「**近道**」
  - エラーメッセージを読まない初心者は上達が遅い

# エディタでのエラー修正

## ■ エディタ(Editor: 編集プログラム)とは

- ソースコードを書くためのソフトウェア(テキストエディタ text editor)
- 「メモ帳」や「ワードパッド」でも可能だが...(不便)

## ■ Visual Studio は統合開発環境

- ソースプログラム完成後、エディタ画面からビルド
- 使用するプログラミング言語に合わせてソースプログラム編集に便宜を図ってくれる
  - ソースファイルの拡張子を .c に設定 例) Myname.c
  - ソースコード編集時のエラー表示(ケアレスミスの低減)
  - { } や ( ) などの括弧の対応, 行末の ; の有無
  - 全て半角英数で記述(スペースには特に注意)
  - 紛らわしい文字や数字に注意 例) 1とl(エル), 0(ゼロ)とo(オー)
  - 適切なインデント(字下げ)の実施
  - (わざと)無視することもできる(警告 warning など)

# エディタでのエラー表示(1)

7

- **わざとエラーを出してみよう!**

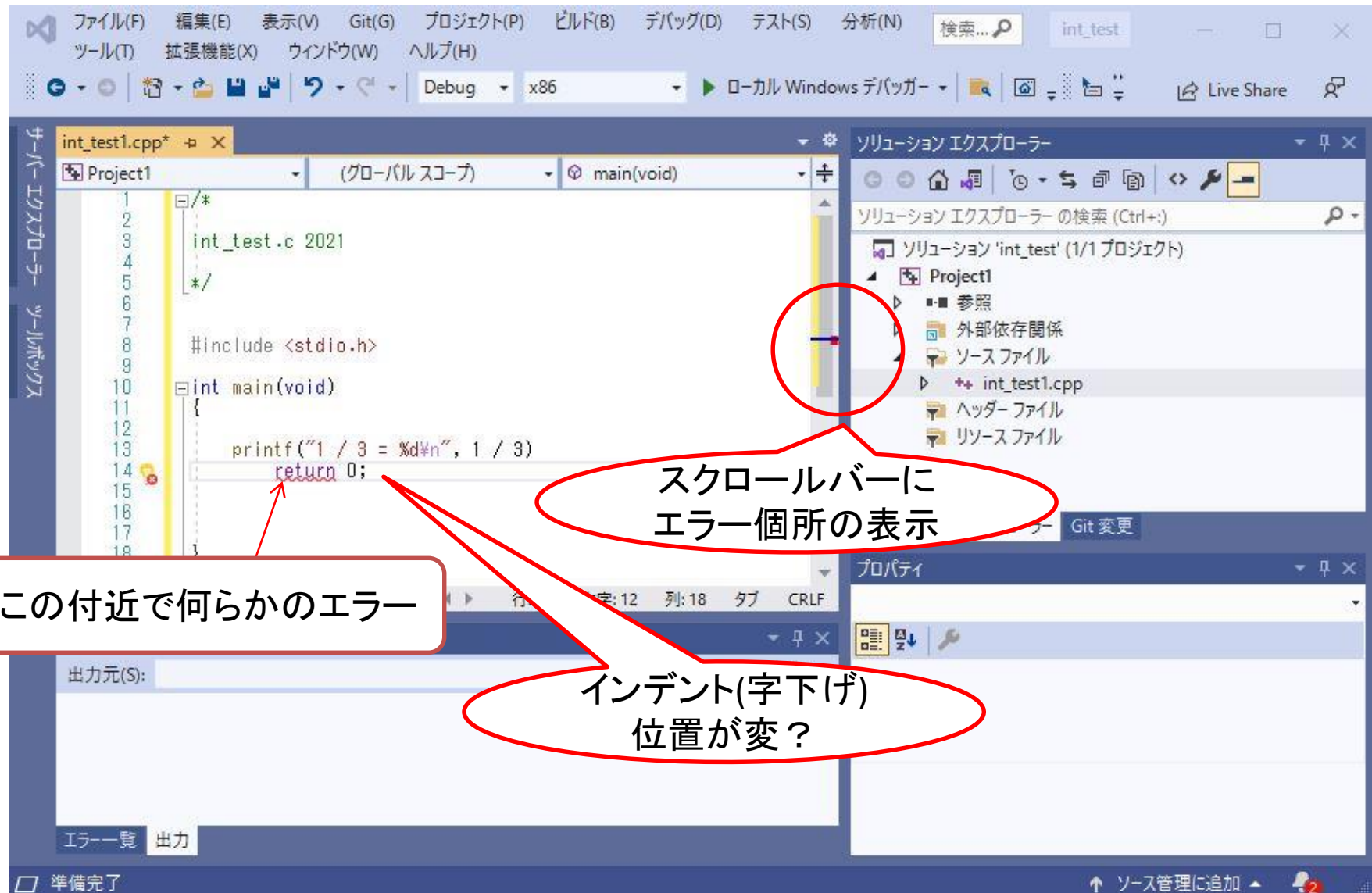
- ソリューション名: int\_test
- プロジェクト名: int\_test1

実際にプログラムを忠実に  
入力して、以降のエラー表示  
を自身で確認すること!

```
/*  
    int_test1.c 2021  
*/  
  
#include <stdio.h>  
  
int main(void)  
{  
  
    printf( " 1/3 = %d¥n", 1/3 )  
    return 0;  
  
}
```

# エディタでのエラー表示(2)

- return に赤いアンダーラインが表示される





# ビルド時: エラーメッセージの表示

- エラーがあるのはわかるが、どんなエラーかわからない
- ビルドを実行してエラーメッセージを表示させる

```
出力
出力元(S): ビルド
ビルドを開始しました...
1>----- ビルド開始: プロジェクト: Project1, 構成: Debug Win32 -----
1>int_test1.cpp
1>C:\Users\Nakano\source\int_test\Project1\int_test1.cpp(14,3): error C2143: 構文エラー: ';' が 'r
1>プロジェクト "Project1.vcxproj" のビルドが終了しました -- 失敗。
===== ビルド: 0 正常終了、1 失敗、0 更新不要、0 スキップ =====
```

「エラー一覧」  
タブ

エラーがあると、「失敗」の前  
にエラーの数が表示される

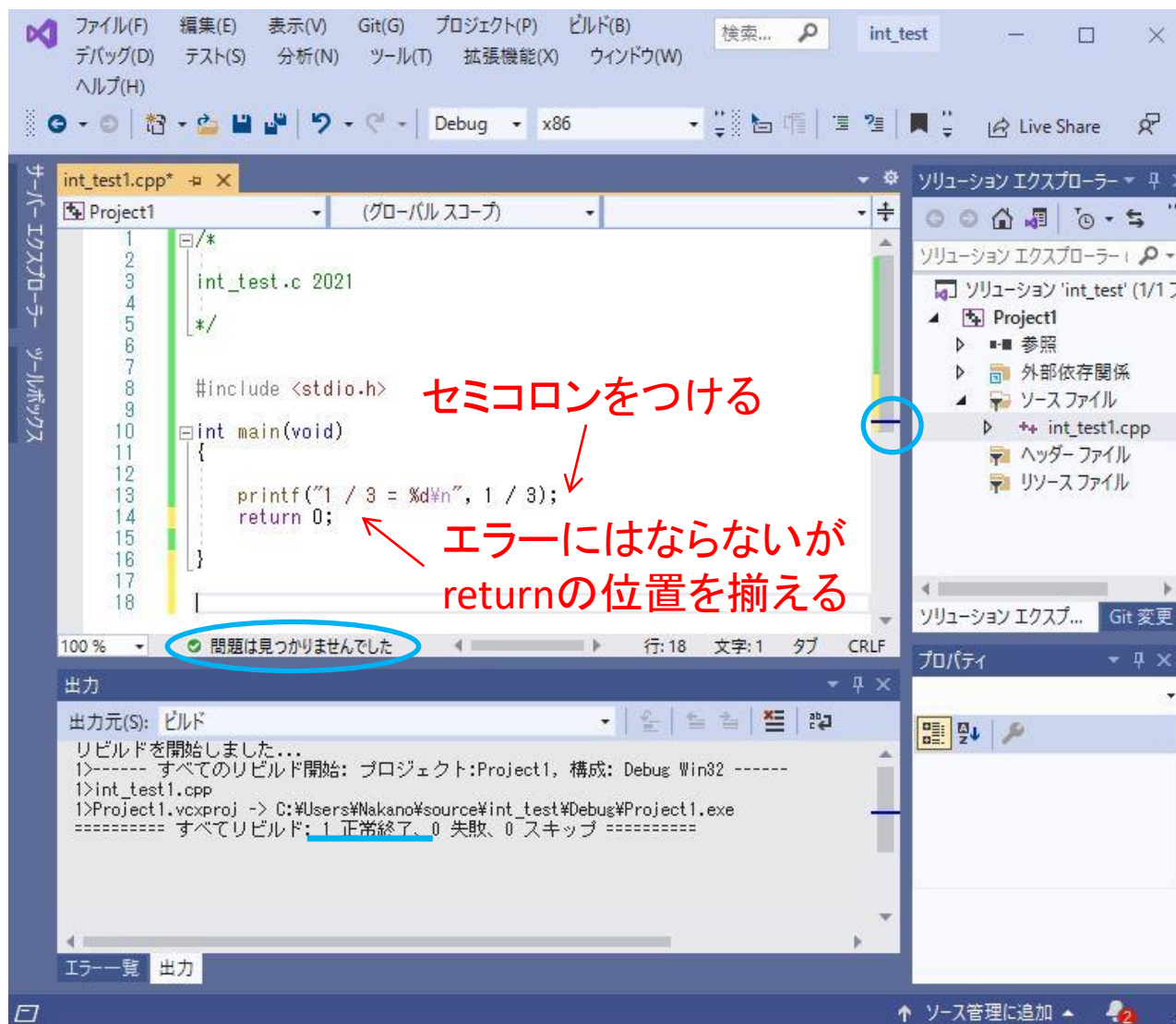
	コード	説明	プロジェクト	
✖	C2143	構文エラー: ';' が 'return' の前にありません。	Project1	int_test1.cpp 14
abc	E0065	';' が必要です	Project1	int_test1.cpp 14

エラー項目をダブルク  
リックすると該当行をエ  
ディタが表示する

構文エラー: ';' が 'return' の前にありません

# エラーの修正

; (セミコロン)を(前の行の終わりに)付けてみる。



# エラーの修正

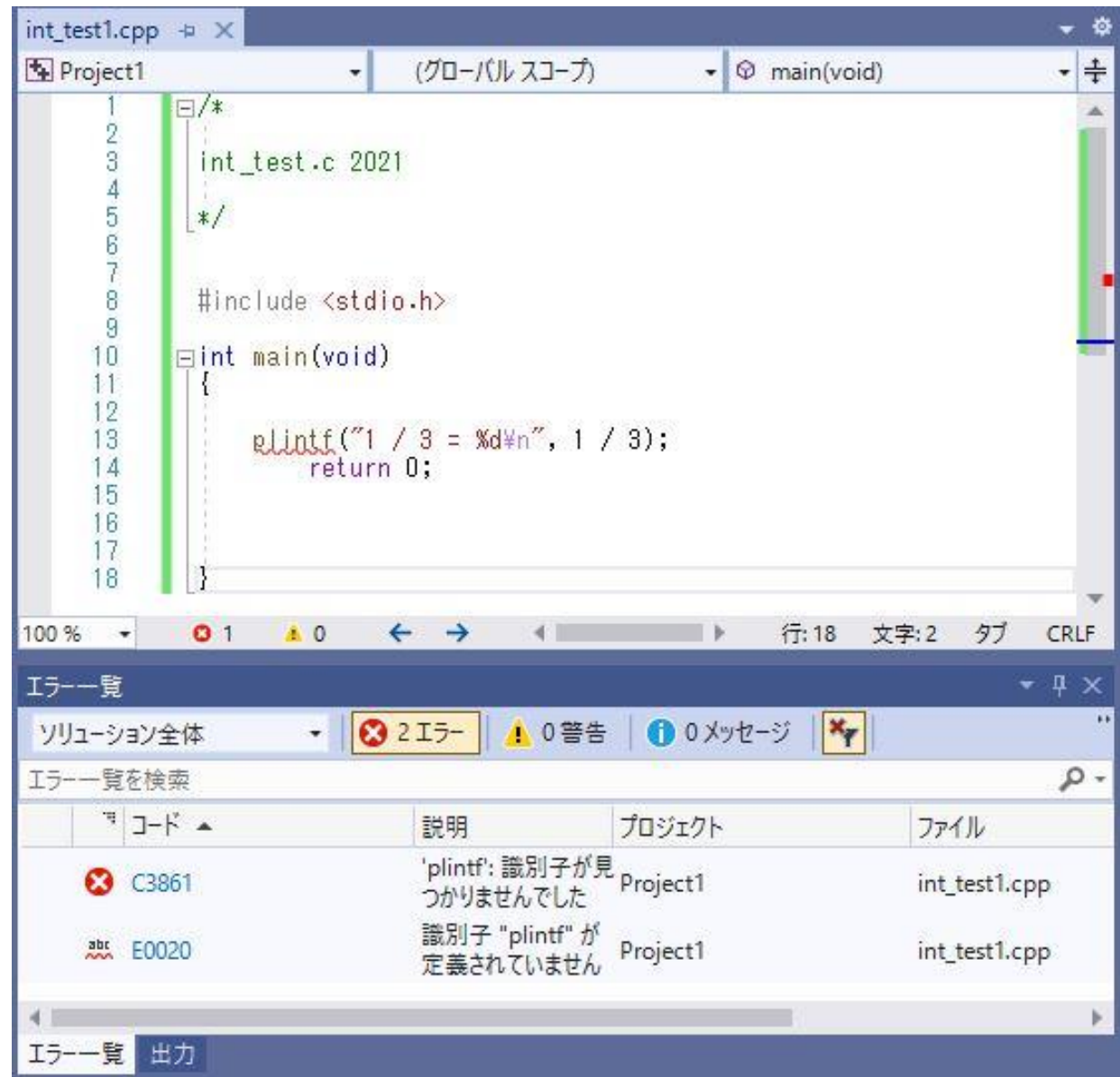
11

- 関数名を間違えると...

printf ->  
plntf

- ビルドすると次のエラーが表示

「'plntf':識別子が見つかりませんでした」



# エラーメッセージがなくなったからといって、正しいプログラムができたというわけではない

```
/*  
    int_test1.c    2021  
*/  
  
#include <stdio.h>  
  
int  main( void )  
{  
  
    printf( " 1 / 3 = %d¥n", 1 / 3 ) ;  
    return 0 ;  
  
}
```

エラーやウォーニング(警告)なく、ビルドできるが....

整数演算の結果	$1/3 = 0$
実数演算の結果	$1/3 = 0.3333333$

プログラム作成者はどちらを意図しているのか？

# 結果は？

```
C:\Users\Nakano\source\int_test\Debug>Project1
1 / 3 = 0

C:\Users\Nakano\source\int_test\Debug>
```

- $1 / 3 = 0$  整数演算が**正しく**行われている
- $0.333333$  が表示されると期待していた場合にはこの結果は**誤り**である
  - では  $0.333333$  を表示させるにはどうすれば良いのか？
- エラーメッセージが出ないけれども、**正しいプログラムが作れたとは限らない！**
  - プログラム初心者が陥りやすい。結果をよく確認することが大切

# 数値、変数の種類と区別

- 数値の区別と表示
  - 整数と実数
  - 実数の精度
  - `printf()` 中の表示形式(`%d` `%f` `%s`)
- 変数について
  - 変数名の付け方
  - 変数宣言
    - `int` `float` `double`

# 数値、変数の種類と区別

- 数値の区別と表示
  - 整数と実数
  - 実数の精度
  - `printf()` 中の表示形式(`%d` `%f` `%s`)
- 変数について
  - 変数名の付け方
  - 変数宣言
    - `int` `float` `double`

# 整数と実数

- 数値は「整数」と「実数」に分けられる
  - 整数      1    2045      -5896832    (3ケタごとの, を付けない)
  - 実数    1.0   2045.0    -5.896382e+6 (小数点は . (dot ))
    - -5.896382e+6は  $-5.896382 \times 10^{+6}$ を示している
    - 0.00256 と 2.56e-3 は同じ数でどちらの表記もOK
- 整数の使用用途(最後の桁まで誤差が生じない)
  - 順番, 番号
  - 金額
- 実数の使用用途(精度が高いが, 誤差を生じる場合も)
  - 寸法や重量など
  - 科学で使われる定数(円周率, 指数など)



# 整数

- 整数は4種類 扱える数値の範囲が異なる
  - 整数 `int` 4バイト (32bit)
  - 短整数 `short int` 2バイト (16bit)
  - 長整数 `long int` 4バイト (32bit)
  - 長長整数 `long long int` 8バイト (64bit)
- `unsigned` をこれらの定義の前に付けて「正の整数」
  - 正確には「0(ゼロ)を含む正の整数」 順番や番号に使われる
    - `unsigned int`
    - `unsigned long int`
- どれを使うべきか？
  - 通常は整数(`int`)を使っていれば特に大きな問題はない
  - 小規模のマイコンなど、メモリ容量に制限が厳しい場合には `short int` を使うことも考えられる

# 実数

- 実数には2種類
  - 単精度実数      `float`      4バイト(32bit)
  - 倍精度実数      `double`      8バイト(64bit)
  - 扱う実数の精度(桁数)と範囲が異なる
- 通常PCでは `double` を使う方が問題が少ない
  - 小規模のマイコンなどではメモリ容量の制限と演算能力の点から `float` を使うこともある
- 本演習では通常は `double` を使う. 今回に限り `float` を使う
  - `float` では十分な精度が得られないことがある(6桁程度)
  - C言語では `float` より `double` を使うことが推奨されている
  - `double` は情報量が多い(64bit)にもかかわらず, `float` より演算速度が速いことが多い(特にPC)

# さて、問題のプログラム

- どこが悪いのか？

```
/*  
    int_test1.c    2021  
*/  
  
#include <stdio.h>  
  
int  main( void )  
{  
  
    printf( " 1 / 3 = %d¥n", 1 / 3 ) ;  
    return 0 ;  
  
}
```

# printf() 内の変換仕様

```
printf( " 1 / 3 = %d ¥n", 1 / 3 ) ;
```

- 教科書 p.78, 89
  - 整数の場合 `%d`
  - 少数(実数)の場合 `%f`
    - とりあえず数についてはこの2つを覚える

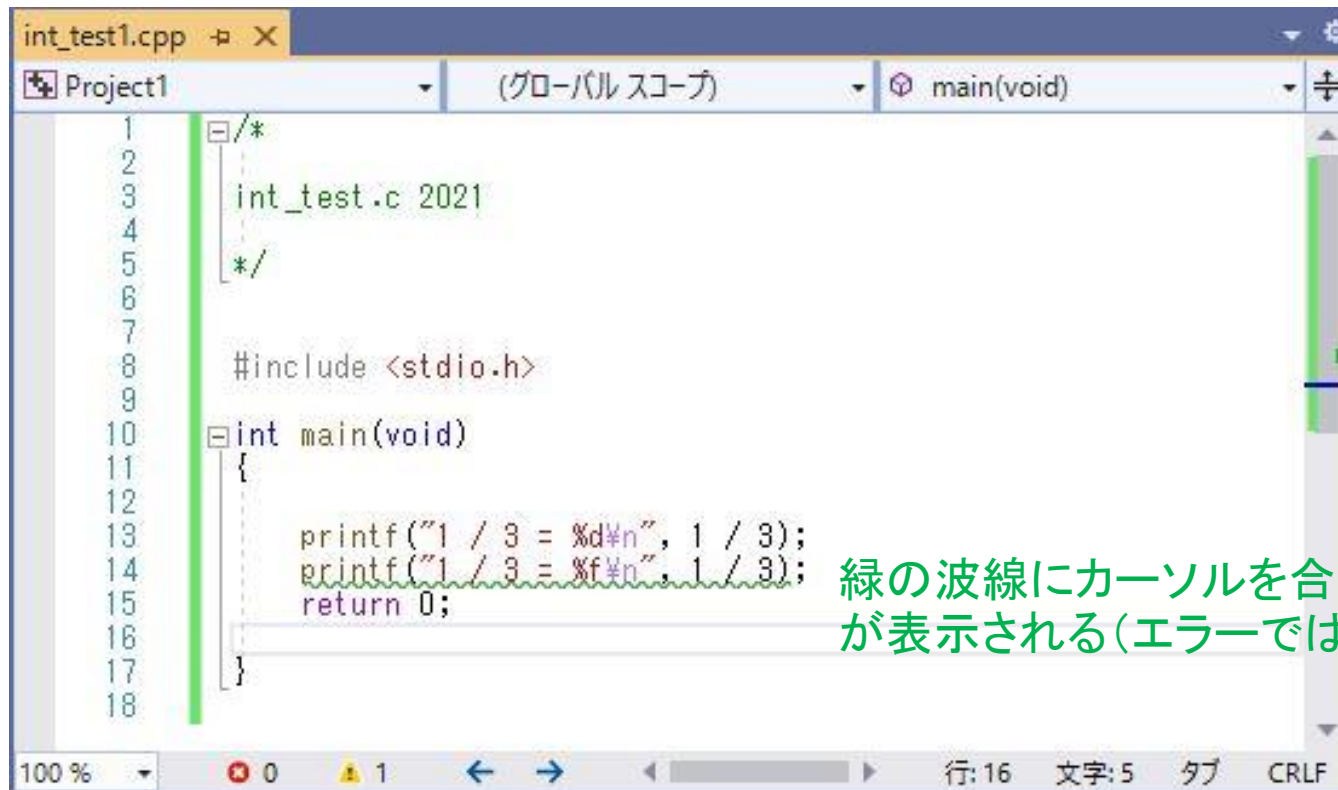
- 次の文を加えて比較してみる

```
printf( " 1 / 3 = %f ¥n", 1 / 3 ) ;
```

# まだ表示したい結果が得られない

21

```
printf( " 1 / 3 = %f ¥n", 1 / 3 ) ;
```



```
int_test1.cpp x
Project1 (グローバル スコープ) main(void)
1  /*
2
3   int_test.c 2021
4
5   */
6
7
8   #include <stdio.h>
9
10  int main(void)
11  {
12
13   printf("1 / 3 = %d¥n", 1 / 3);
14   printf("1 / 3 = %f¥n", 1 / 3);
15   return 0;
16
17  }
18
```

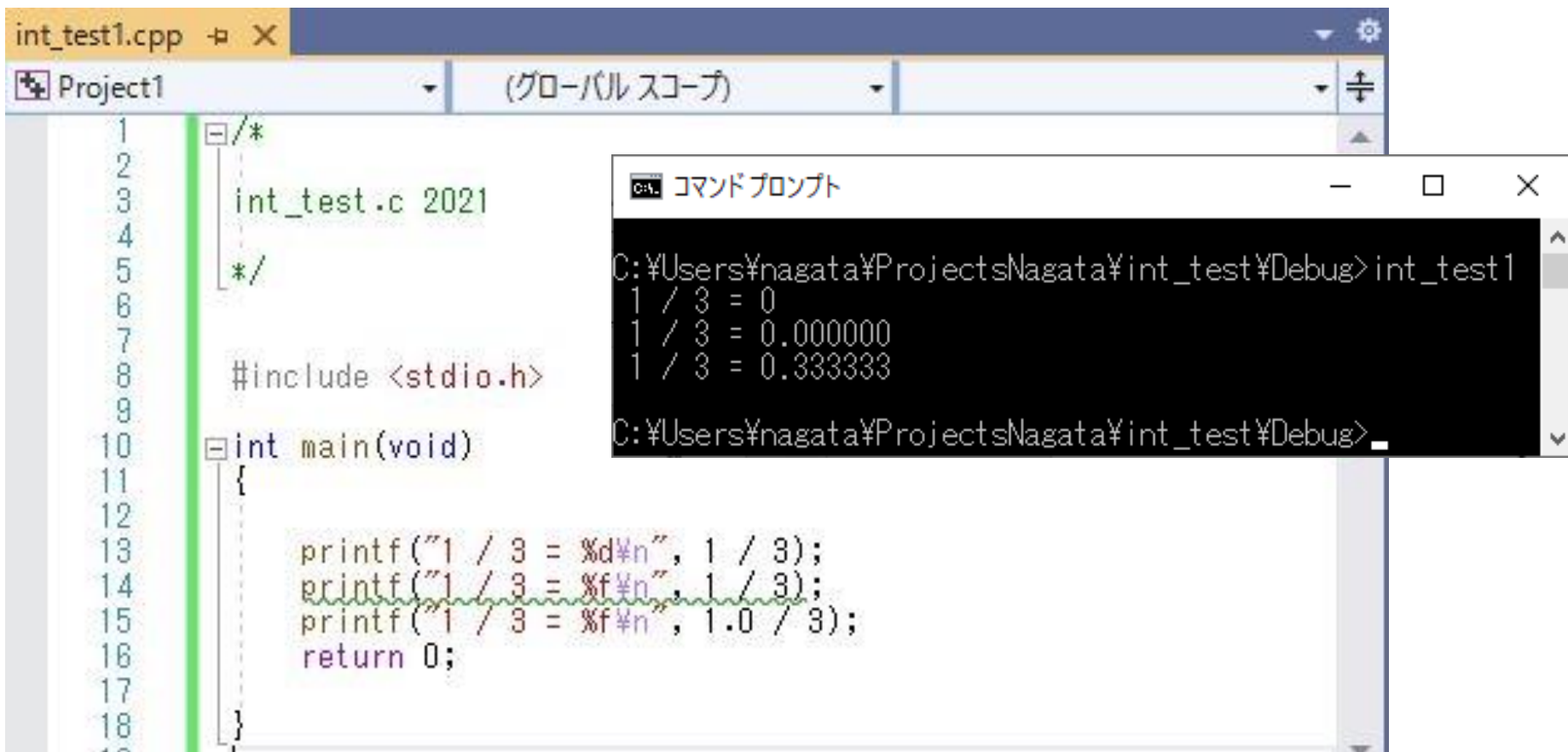
100 % 0 1 行: 16 文字: 5 タブ CRLF

緑の波線にカーソルを合わせると注意が表示される(エラーではない)

```
C:¥Users¥Nakano¥source¥int_test¥Debug>Project1
1 / 3 = 0
1 / 3 = 0.000000
C:¥Users¥Nakano¥source¥int_test¥Debug>
```

# これならOK!

```
printf( " 1 / 3 = %f ¥n", 1.0 / 3 ) ;
```



The screenshot shows a C++ IDE with a file named `int_test1.cpp`. The code is as follows:

```
1  /*
2
3  int_test.c 2021
4
5  */
6
7
8  #include <stdio.h>
9
10 int main(void)
11 {
12
13     printf("1 / 3 = %d¥n", 1 / 3);
14     printf("1 / 3 = %f¥n", 1 / 3);
15     printf("1 / 3 = %f¥n", 1.0 / 3);
16     return 0;
17 }
18
```

Overlaid on the IDE is a Windows Command Prompt window titled "コマンドプロンプト". It shows the execution of the program:

```
C:¥Users¥nagata¥ProjectsNagata¥int_test¥Debug>int_test1
1 / 3 = 0
1 / 3 = 0.000000
1 / 3 = 0.333333
C:¥Users¥nagata¥ProjectsNagata¥int_test¥Debug>_
```

- 1/3 の 1 も 3 も整数なので、演算結果は整数として扱われる
- %f を使って表示を変えても演算結果は変わらない

# これならOK!

```
printf( " 1 / 3 = %f ¥n", 1.0 / 3 ) ;  
printf( " 1 / 3 = %f ¥n", 1 / 3. ) ;
```

The screenshot shows a C++ IDE with a file named `int_test1.cpp`. The code is as follows:

```
1  /*  
2  
3  int_test.c 2021  
4  
5  */  
6  
7  
8  #include <stdio.h>  
9  
10 int main(void)  
11 {  
12  
13     printf("1 / 3 = %d¥n", 1 / 3);  
14     printf("1 / 3 = %f¥n", 1 / 3);  
15     printf("1 / 3 = %f¥n", 1.0 / 3);  
16     printf("1 / 3 = %f¥n", 1 / 3.);  
17  
18     return 0;  
19  
20 }  
21
```

Overlaid on the IDE is a Windows Command Prompt window titled "コマンドプロンプト". It shows the command `Project1` being executed, resulting in the following output:

```
C:¥Users¥Nakano¥source¥int_test¥Debug>Project1  
1 / 3 = 0  
1 / 3 = 0.000000  
1 / 3 = 0.333333  
1 / 3 = 0.333333  
C:¥Users¥Nakano¥source¥int_test¥Debug>
```

- 1 / 3 の 1 も 3 も整数なので、演算結果は整数として扱われる
- %f を使って表示を変えても演算結果は変わらない

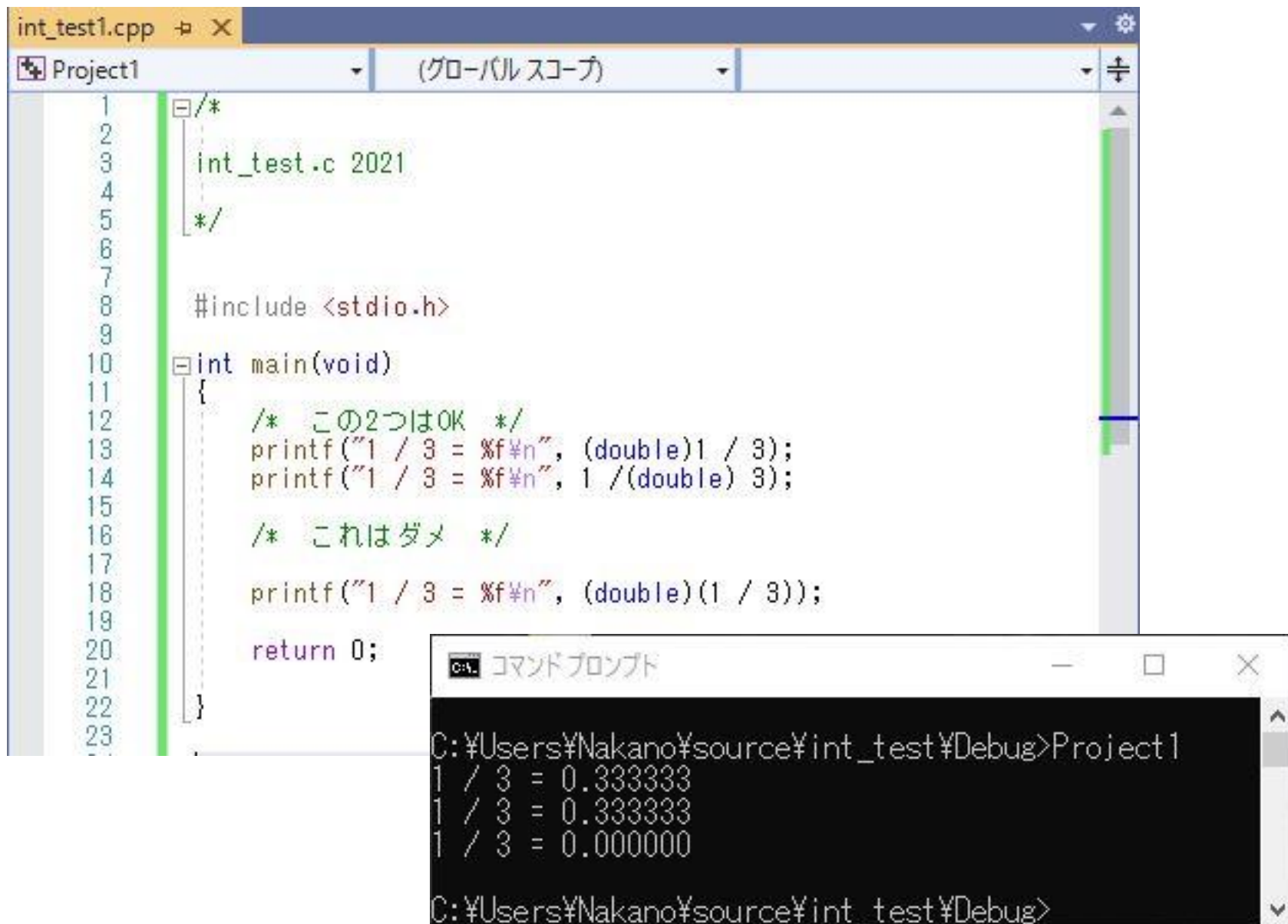
# キャスト(明示的型変換)

- 変数や数の前に ( 型名 ) と付けることによって強制的に型変換することができる
- ここでは1や3の整数を (double) で倍精度実数にしている

```
printf(" 1 / 3 = %f¥n", (double) 1 / 3 );  
printf(" 1 / 3 = %f¥n", 1 / (double) 3 );
```



# キャストの実行結果



```
int_test1.cpp  X
Project1 (グローバルスコープ)
1  /*
2
3  int_test.c 2021
4
5  */
6
7
8  #include <stdio.h>
9
10 int main(void)
11 {
12     /* この2つはOK */
13     printf("1 / 3 = %f\n", (double)1 / 3);
14     printf("1 / 3 = %f\n", 1 / (double) 3);
15
16     /* これはダメ */
17
18     printf("1 / 3 = %f\n", (double)(1 / 3));
19
20     return 0;
21 }
22
23
```

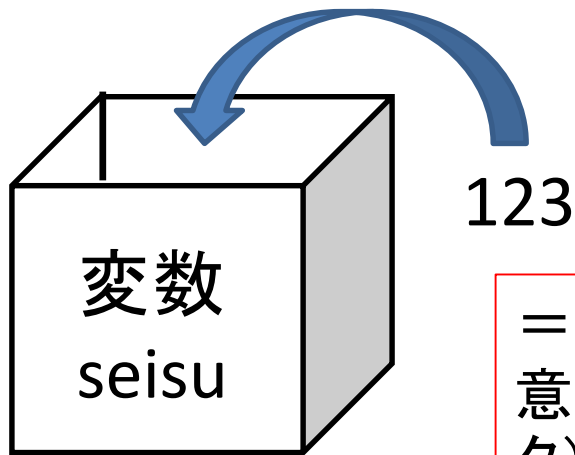
```
コマンドプロンプト
C:\Users\Nakano\source\int_test\Debug>Project1
1 / 3 = 0.333333
1 / 3 = 0.333333
1 / 3 = 0.000000
C:\Users\Nakano\source\int_test\Debug>
```

# 数値、変数の種類と区別

- 数値の区別と表示
  - 整数と実数
  - 実数の精度
  - `printf()` 中の表示形式(`%d` `%f` `%s`)
- 変数について
  - 変数名の付け方
  - 変数宣言
    - `int` `float` `double`

# 変数とは

数学にも変数と呼ばれる概念があるが、プログラミングにおける変数は“**色々な種類の値を入れる(代入する)ことができる箱**”である



例)

```
int seisu;    変数の生成  
seisu = 123;  値の代入
```

= (イコール) は数学では両辺が等しいという意味になるが、プログラミングでは右辺(データ)を左辺(変数)に**代入する**という意味になる

変数に入れることができる値としては、

- 整数 (1, 2, 3, )
- 実数 (1.23456)
- 文字列 (“abcdef”)

# 変数のルール

変数を用いるには以下のルールが存在する

- 変数を使用(生成)すると宣言する(変数宣言)
- どの種類の値を入れる変数なのかを指定する(変数の型)
- 変数は名前(変数名)を付ける
- 「変数=値;」のイコールは両辺が等しいということではなく、値を変数に代入することを意味する

# 変数の宣言

- 変数を使用するには、変数の型、変数名を宣言

変数の型 変数名;

例) int seisu;  
double syosu;

- 同じ変数の型の変数名は同時に複数宣言可能

変数の型 変数1, 変数2, 変数3;

例) int a, b, c;

変数名はその変数の役割等が分かるように簡潔に付けるのがよい

# 変数の型

値の種類	変数の型	名称	ビット長	値の範囲
型なし	void	void型	無し	無し
整数 []は省略可能	[signed] short int	(符号付き)短整数型	16	-32768 ~ 32767
	unsigned short int	符号なし短整数型	16	0 ~ 65535
	[signed] int	(符号付き)整数型	32	-2147483648 ~ 2147483647
	unsigned int	符号なし整数型	32	0 ~ 4294967295
	[signed]long int	(符号付き)長整数型	32	-2147483648 ~ 2147483647
	unsigned ling int	符号なし長整数型	32	0 ~ 4294967295
	[signed]long long int	(符号付き)長長整数型	64	-9223372036854775808 ~ 9223372036854775807
	unsigned long long int	符号なし長長整数型	64	0 ~ 18446744073709551615
浮動小数点	float	単精度実浮動小数点型	32	Min: 1.175494351e-38 Max: 3.402823466e+38
	double	倍精度実浮動小数点型	64	Min: 2.2250738585072014e-308 Max: 1.7976931348623158e+308
文字列	[singed] char	符号付き 文字型	8	-128 ~ 127
	[unsigned] char	符号なし 文字型	8	0 ~ 255

# 変数へ値の代入

- 変数を宣言すると同時に値を入れることができる  
(初期化)

変数の型 変数名 = 値;

例) `int num = 0;`

- 変数に入っていた値に加算される

変数 = 変数 + 1;

例) `int num1 = 1;`  
`num1 = num1 + 1;`

# 変数を用いた演算

- 計算の結果を変数に代入

```
num1 = 1 + 3;
```

- 変数同士の演算を変数に代入

```
int num1 = 2;
```

```
int num2 = 3;
```

```
int num3 = 0;
```

```
num3 = num1 + num2;
```



# 変数名のつけ方(1)

## エラーにならない変数名

- アルファベットと数字を使う (半角英数)
  - 記号やスペースは使えない
    - 例外 `_` (アンダースコア)
  - 第1文字は必ずアルファベット
  - C言語の文法ですでに使われている名前(予約語)は使ってはいけない
    - (OK) `a_and_b`, `alpha`, `beta2`, `miyazaki`,
    - (NG) `a b`, `a+b`, `3nensei`, `printf`

# 変数名のつけ方(2)

## エラーにはならないが...

- 大文字と小文字は区別される
- 基本的に小文字にする
- 簡潔でわかりやすい変数名を付ける
  - 使い捨ての変数でない限り, 変数の意味を名前に付ける
  - 英語でなくローマ字でも可
    - (良くない例) a, b, x, y, z (一文字)
    - (良い例) value, sum1, max, goukei, shouhizei

# 変数名のつけ方(3)

- 変数名に 整数/実数の区別を付ける
  - (慣例的に) 整数変数を i, j, k, l, m, n で始まる変数名にする
    - number, num1, iter, member ....
    - ivalue, nfiles, nrow, ncol, ....
  - 変数名の最初に i (int), f (float), d (double) を付ける
    - iClassNumber, fHeight, dWeight,
    - 変数名にスペースが使えないため, 変数名に2つ以上の単語をつなげる場合, 単語の先頭の文字だけ大文字にする