

モジュール分割

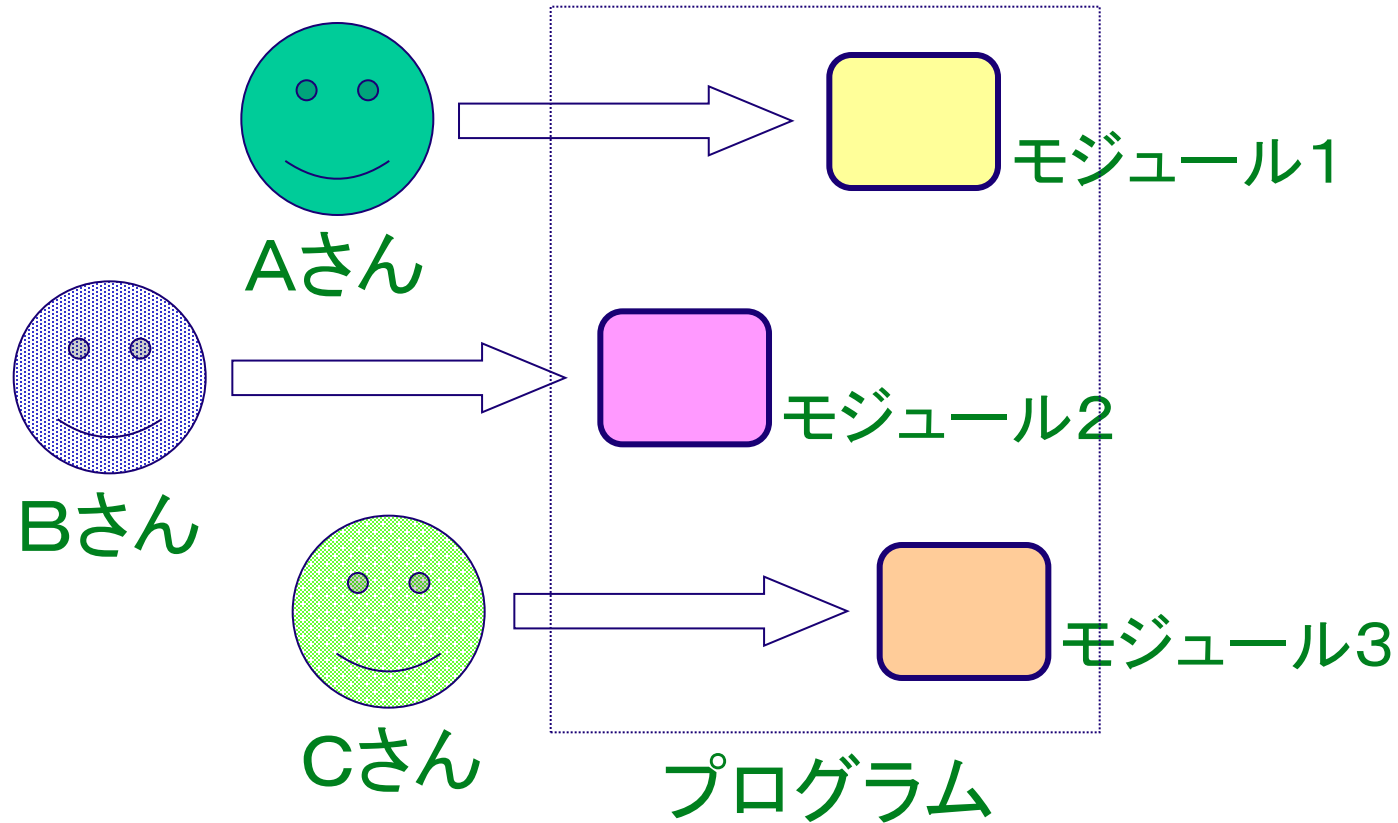
大規模なプログラム

- プログラムは、多数の関数の集まり
- モジュール分割が必要
 - すべてのプログラムを1つのファイルに入れるのではなく、複数のファイル(モジュール)に分けて格納

あらすじ

- プログラムを「モジュール」に分割することの利点
- 他のモジュール内の「関数」を呼び出す方法

モジュール分割の利点



- プログラムの共同開発がしやすい
- 関連の深い関数を、同じモジュールに集まることで、プログラム全体の見通しがよくなる

例題2. プログラムのファイル分割

- ファイルの中身を表示するプログラムを作る
- 役割ごとに, 関数を作る.
 1. ファイル名の読み込み
 2. ファイルのオープン
 3. ファイルの読み込み
 4. ファイルのクローズ
- プログラムは, 3つのファイル `main.cpp`, `input.cpp`, `file.cpp` に分割する

ファイル分割の例

関数 main

ファイル main.cpp

関数 get_file_name

ファイル input.cpp

関数 open_file

関数 print_file

関数 close_file

ファイル file.cpp

- ファイルには、いくつかの関数(1つ以上)を入れる

プログラム例

ファイル名の読み込み



ファイルのオープン



ファイルの表示



ファイルのクローズ

ファイル名の読み込み

```
void input_file_name( char* file_name )
{
    char *buffer[80];
    printf( "ファイル名をどうぞ？¥n" );
    fgets( buffer, 80, stdin );
    sscanf( buffer, "%s", file_name );
}
```


ファイルのオープン

```
FILE* file_open( char* file_name )
{
    FILE* in_file = fopen( file_name, "r" );
    if ( in_file == NULL ) {
        printf( "ファイルオープンに失敗しました");
    }
    return in_file;
}
```

ファイルの表示

```
void file_open( FILE* in_file )
{
    char *buffer[100];
    lseek( in_file, LEEK_SET );
    while ( fgets( buffer, 100, in_file ) != NULL ) {
        printf( "%s¥n", buffer );
    }
}
```

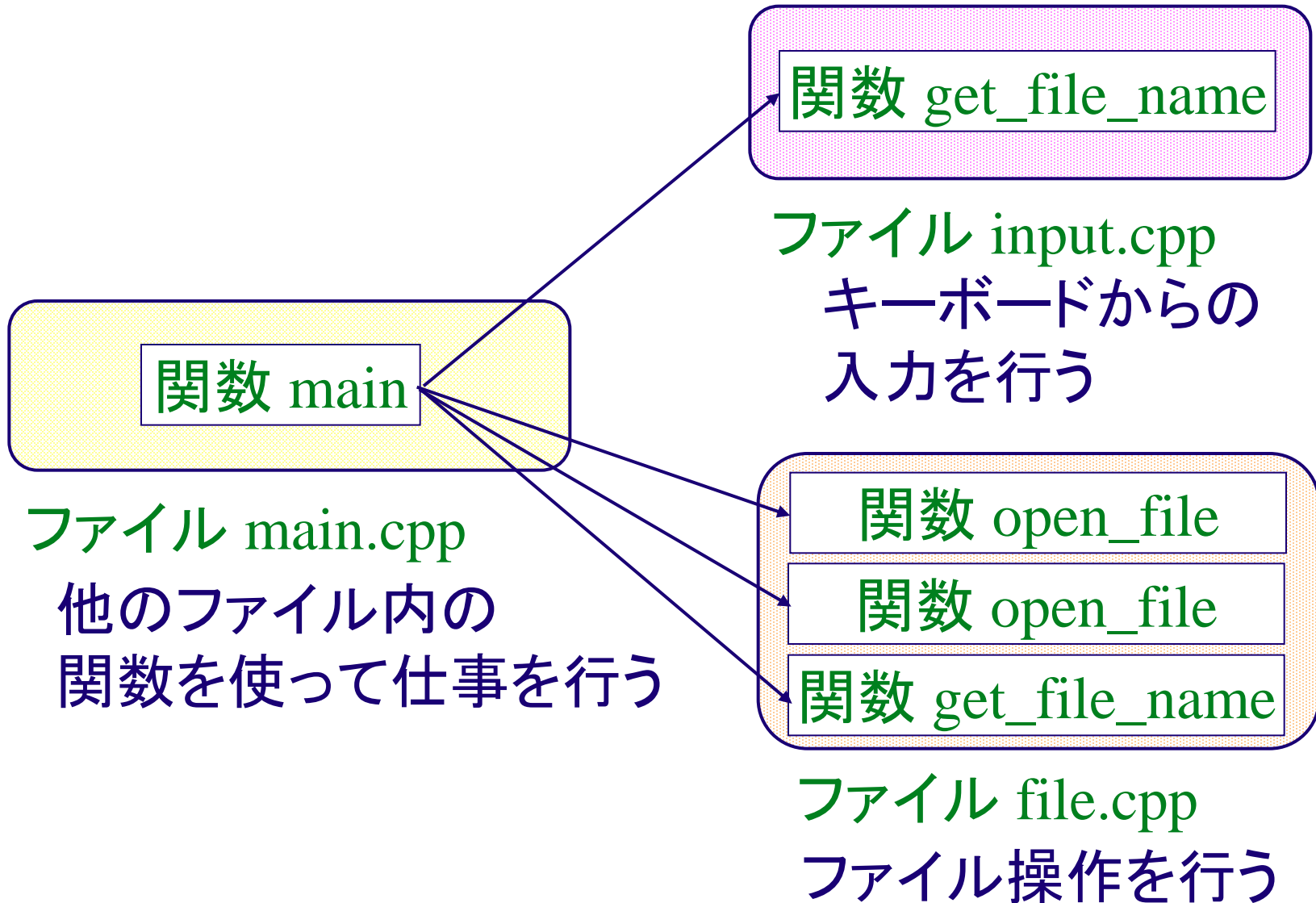
ファイルのクローズ

```
void file_close( FILE* in_file ) {  
    fclose( in_file );  
}
```

プログラムと関数呼び出し

```
#include <stdio.h>
int main()
{
    char file_name[80];
    get_file_name( file_name );
    open_file( file_name );
    print_file();
    close_file();
}
```

関数の関係



モジュールの意味

- 機能のまとめり
- 呼び出し呼び出されの関係
- 他のモジュール内の「関数」を呼び出す必要あり
- ローカルな変数と共有変数

main モジュールの例

```
#include <stdio.h>
extern void inc_counter();
extern void print_counter();
```

関数 int_counter の使用を宣言
(「extern」は、他のモジュール
にある関数を使いたい」という
意味)

```
int main() {
    int i;
    for ( i = 0; i < 10; i++ ) {
        inc_counter();
    }
    print_counter();
}
```

関数 int_counter の呼び出し

count モジュールの例

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

```
static int counter = 0;
```

変数 counter の宣言

(「static」は「変数はこのモジュール
の中でのみ使う」という意味)

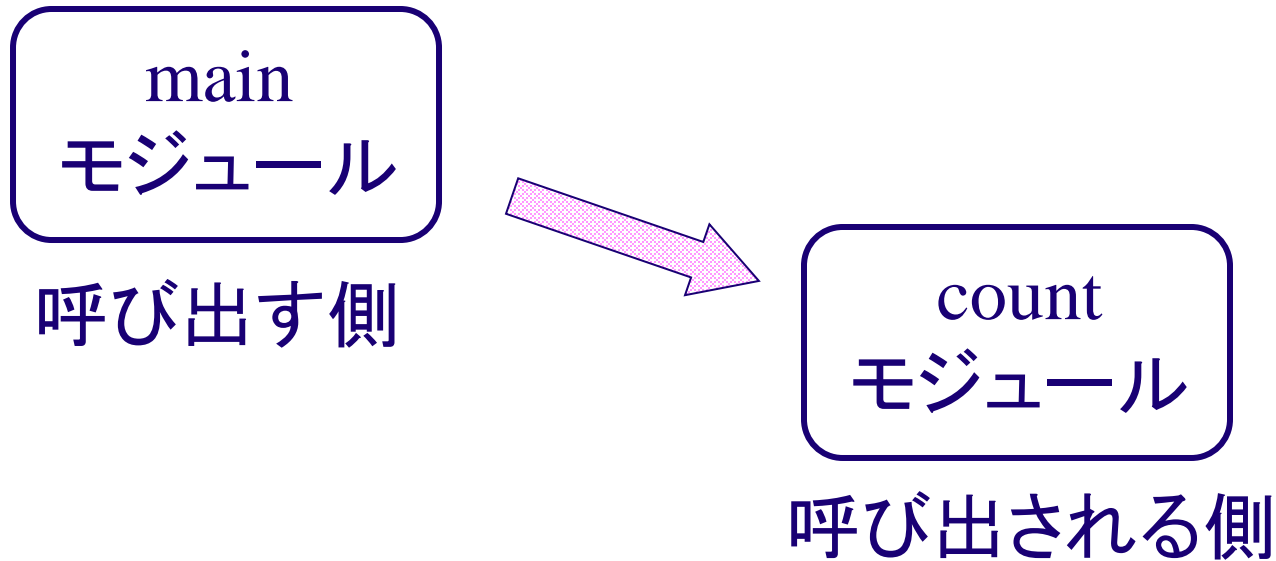
```
void inc_counter() {  
    counter = counter + 1;  
}
```

関数 int counter の実体

(実は、関数は、他の
モジュールからも使うことができる)

```
void print_counter() {  
    printf( "counter is %d¥n", counter );  
}
```


モジュールの意味

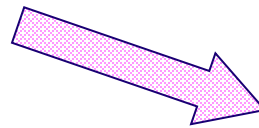


- ファイル `count.cpp`
「数を数える」機能
- ファイル `main.cpp`
`count` モジュールを使って、実際の仕事を行う

main モジュールと count モジュール

main モジュール
count モジュール内
の数を数える関数
inc_counter を使
いたい

ファイル名: main.cpp



count モジュール
関数 inc_couner の実体
は実はここ

ファイル名: count.cpp

- ふつう, 1モジュールは, 1つのファイル
- main モジュールと count モジュールは, 呼び出し・呼び出されの関係

「static」+「変数宣言」

static int counter = 0;

static あり (static 変数)

変数を, このモジュールの中でのみ使う場合

int counter = 0;

static なし (global 変数)

変数を, 他のモジュールから使うかも知れない場合

C++では, 「モジュール」ごとに, 好きな変数を自由に使えるので, プログラムしやすい

変数の有効範囲

- static 変数: モジュールの中だけ

(例) `static int counter = 0;`

- global 変数: 他のモジュールからも使える

(例) `int age = 0; /* static なし */`

- 他のモジュールの変数を直接使うと、結局、変数がどこで書き変わったか分からなくなってしまう
- 「他のモジュールの変数を使う」のはなるべく止めて「他のモジュールの関数を呼び出す」だけにしよう
(global 変数はなるべく使わない)

まとめ

- モジュール分割時の変数と関数の宣言
 - static 変数: モジュールの中でのみ使う変数
- プログラムを「分割」することの意味
 - プログラムが分かりやすくなって、結局は、プログラムを早く、正しく作れるようになる

設問

1. モジュール作成

- ファイル操作を行う次の2つの関数を含むモジュール file.cpp を作成して下さい。ファイルのオープンモードは「r」とせよ。

```
open_file( char *name ); /* ファイルのオープン */  
close_file();           /* ファイルのクローズ */
```

2. extern 文

- 1. で作成したモジュールを使用する main 関数を含む main.cpp を作成して下さい
- main 関数には、「中身の表示」、「行数の表示」など役に立つ機能を実装すること
- file.cpp, main.cpp をコンパイルし、動作を確認して下さい