

第10回関数2

(関数の利用と変数のスコープ)



今回の目標

- voidという型を理解する。
- 関数の副作用について理解する。
- 変数の適用範囲(スコープ)について理解する。
- 多段にわたる関数呼び出しを理解する。

☆階乗を求める関数を利用して、組み合わせの数を求める関数を作成する

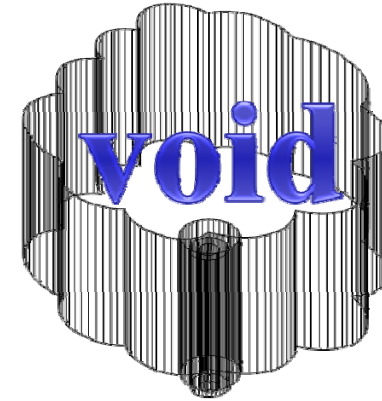
順列の数と組み合わせの数

$${}_nP_m = \frac{n!}{(n-m)!} = \underbrace{n \times (n-1) \times \cdots \times (n-m+1)}_{m\text{個}}$$

$${}_nC_m = \frac{n!}{m! \times (n-m)!} = \frac{\overbrace{n \times (n-1) \times \cdots \times (n-m+1)}^{m\text{個}}}{\underbrace{m \times (m-1) \times \cdots \times 1}_{m\text{個}}}$$

引数が無い関数

仮引数(関数への入力)や、
戻り値(関数からの出力)が無いことを、
voidという型であらわす。



仮引数の無い関数例

括弧だけを記述する。

```
int function1()  
{  
    return 0;  
}
```

明示的にvoidと記述する。

```
doulbe function2(void)  
{  
    return 1.0;  
}
```

戻り値の無い関数

戻り値の無い関数例

明示的にvoidと記述する。

```
void function3(int a)
{
    return;
}
```

return の後に式を書かない。

関数の副作用

仮引数や戻り値以外の動作を副作用という。
標準入出力への値のやり取り等が代表的な副作用である。
仮引数や戻り値が無い関数でも、副作用によって処理を行える。
(数学的な関数と大幅に異なる機能である。)

副作用例:

```
int input(void)
{
    scanf("%d",&a);
    return a;
}
```

標準入力から値を
読み込む副作用

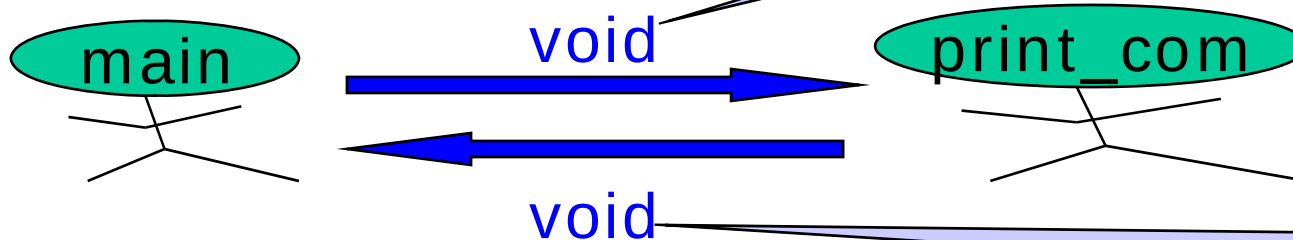
```
void kaigyo(void )
{
    printf("¥n");
    return;
}
```

標準出力へ値を
出力する副作用

練習1

```
/*test_void.c 練習1コメント省略*/  
#include<stdio.h>  
void print_com(void);  
int main()  
{  
    print_com( );  
    return 0;  
}  
  
void print_com(void)  
{  
    printf("print_com内で実行¥n");  
    return;  
}
```

いつもの
処理やって



終わったよ。

変数のスコープ^o1 (有効範囲1)

関数定義の一般的な書式:

```
型1 関数名1(型1a 仮引数1a, 型1b 仮引数1b, ...)  
{  
    /* 変数宣言 */  
    型x    変数x  
}
```

引数が複数ある場合は、
引数リスト中で
各引数をカンマ「,」
で区切る。

仮引数とその関数内で宣言した変数は、
宣言した関数の内部だけで有効である。

したがって、異なる2つの関数で同じ変数名を用いても、
それぞれの関数内で別々の変数としてあつかわれる。

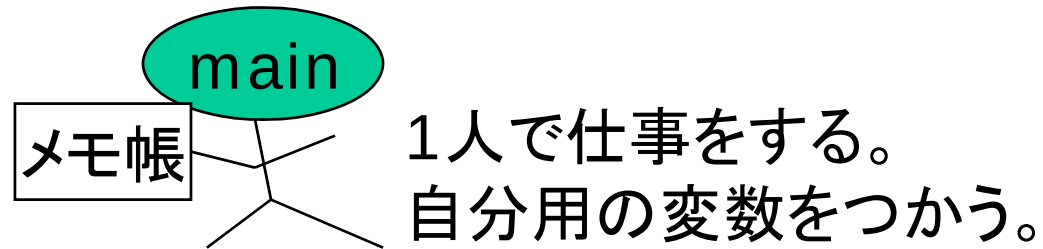
```

int    main()
{
    /* 変数宣言 */
    mainの変数
}
型1    関数1(型1a 仮引数1a,型1b 仮引数1b)
{
    /* 変数宣言 */
    関数1の変数
}
型2    関数2(型2a 仮引数2a,型2b 仮引数2b)
{
    /* 変数宣言 */
    関数2の変数
}

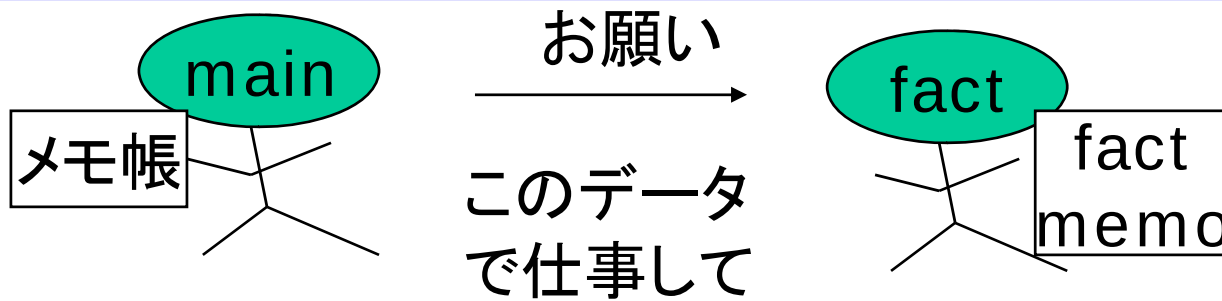
```

イメージ

いままでは、main関数1つしかなかった。



mainとfactがあると

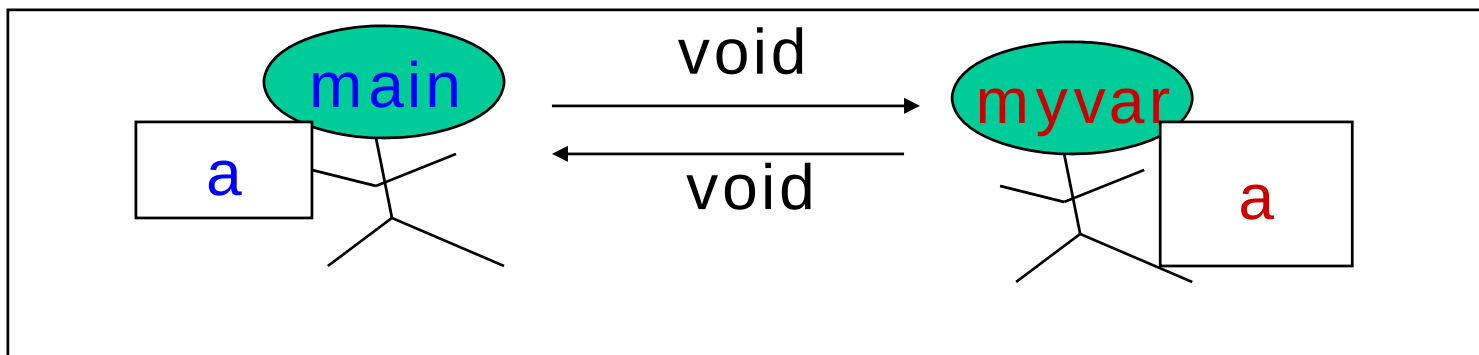


練習2

```
/*test_scope.c 練習2 コメント省略*/
#include<stdio.h>
void myvar(void);
int main()
{
    int a;

    printf("( In main) Input a= ? ");
    scanf("%d",&a);
    printf("(In main) a= %d ¥n",a);
    myvar();
    printf("(In main) a= %d ¥n",a);
    return 0;
}
/* 次が続く */
```

```
/* 続き */  
void myvar(void)  
{  
    int a;  
  
    printf("( In myvar) Input a= ? ");  
    scanf("%d",&a);  
    printf("(In myvar) a= %d ¥n",a);  
  
    return;  
}
```



グローバル変数とローカル変数

実は、関数の外でも、変数の宣言ができます。
その変数をグローバル変数と呼びます。

一般的な形

```
/*グローバル変数宣言*/  
型A    変数A;  
型B    変数B;  
  
int main()  
{  
    /* mainのローカル変数宣言 */  
}
```

本演習では、
グローバル変数は、
1文字だけ英大文字
で残り小文字にしましょう。
(スタイル規則参照)

```
int Global1;
```

グローバル変数のスコープ

```
/*グローバル変数宣言*/
```

グローバル変数

```
int    main()
```

```
{
```

```
/*変数宣言*/
```

mainの変数

```
}
```

```
型1    関数1(型1a 仮引数1a,型1b 仮引数1b)
```

```
{
```

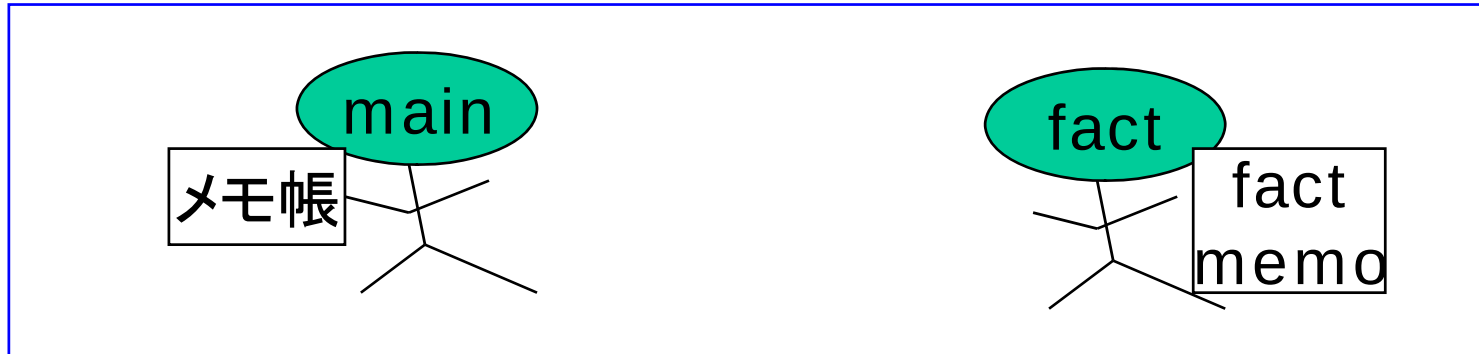
```
/*変数宣言*/
```

関数1の変数

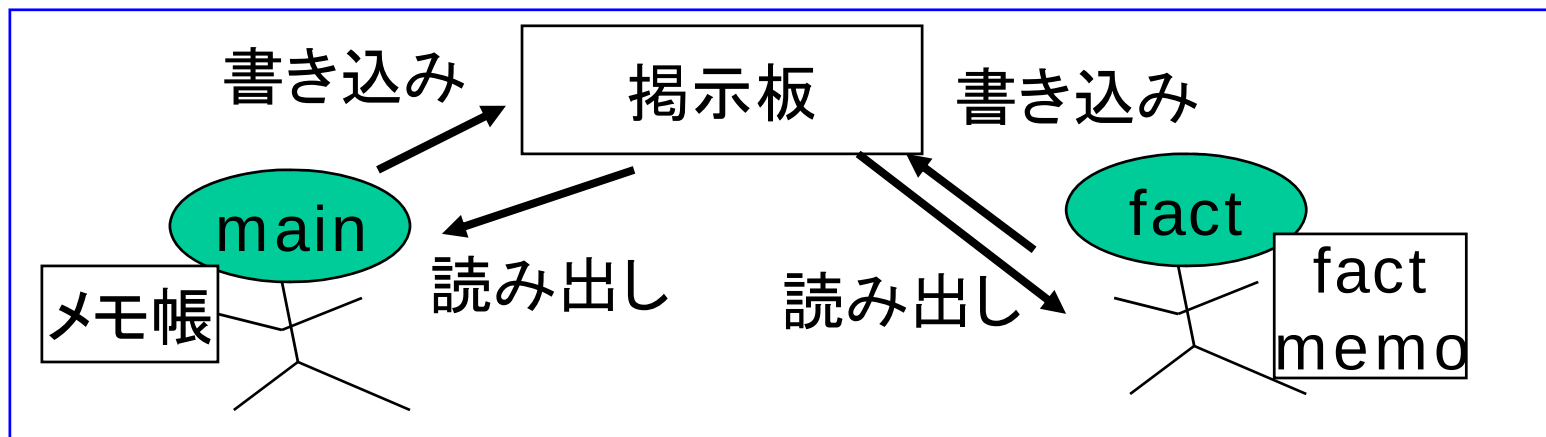
```
}
```

ローカル変数とグローバル変数

ローカル変数は、自分用のメモ帳



グローバル変数は、どの関数でも読み書きできる掲示板



グローバル変数とローカル変数が同じ名前的时候は？

ある関数でグローバル変数と同じ名前の変数を宣言すると、その関数内ではその変数名はローカル変数としてあつかわれる。したがって、グローバル変数の変更はおこなわれない。
(変数名が同じものの同士では、そのスコープが狭いものが優先される。
関数が違えば、同じ変数名でも大丈夫。)

注意：

本演習のスタイルでは、
ローカル変数とグローバル変数は必ず異なる。
ローカル変数：すべて小文字
グローバル変数：1文字目大文字
マクロ名：すべて大文字

多段にわたる関数呼び出し

main関数以外の関数からでも、関数を呼び出せる。

```
int main()
{
    pe=p(m,n);
    return 0;
}
```

```
int p(int n,int m)
{
    pe=f(n)/f(n-m);
    return pe;
}
```

```
int f(int n)
{
    int fa=1.0;
    for(i=1;i<n;i++)
    {
        fa=fa*i;
    }
    return fa;
}
```

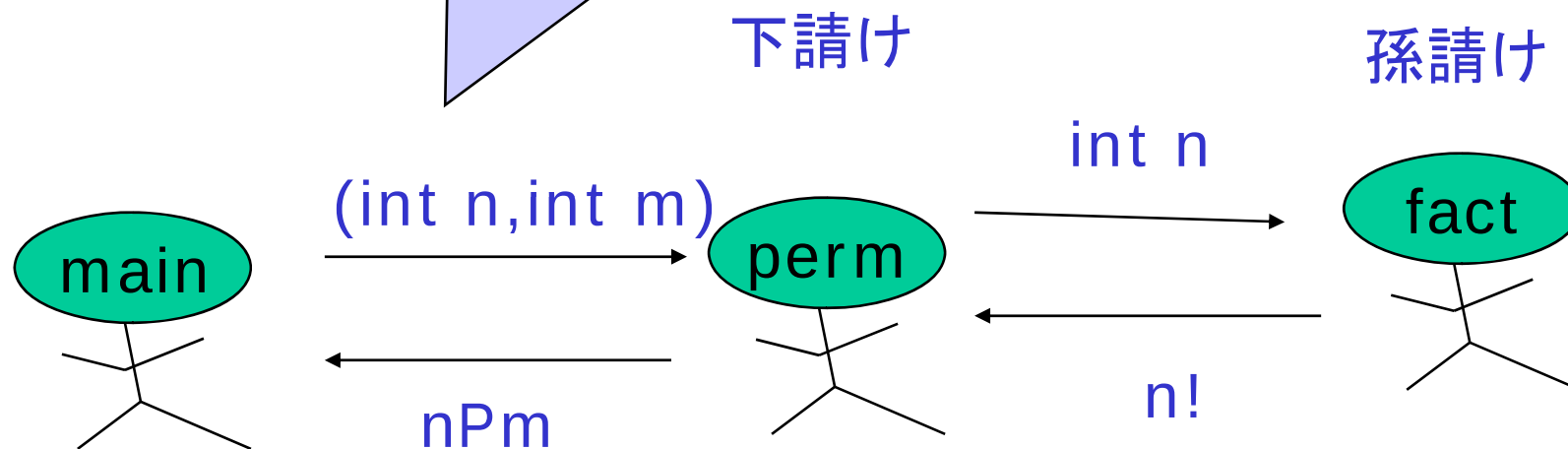
main関数以外からでも、
定義した関数を呼び出
せる。

main関数でのreturn文でプロ
グラム全体が終了する。

イメージ

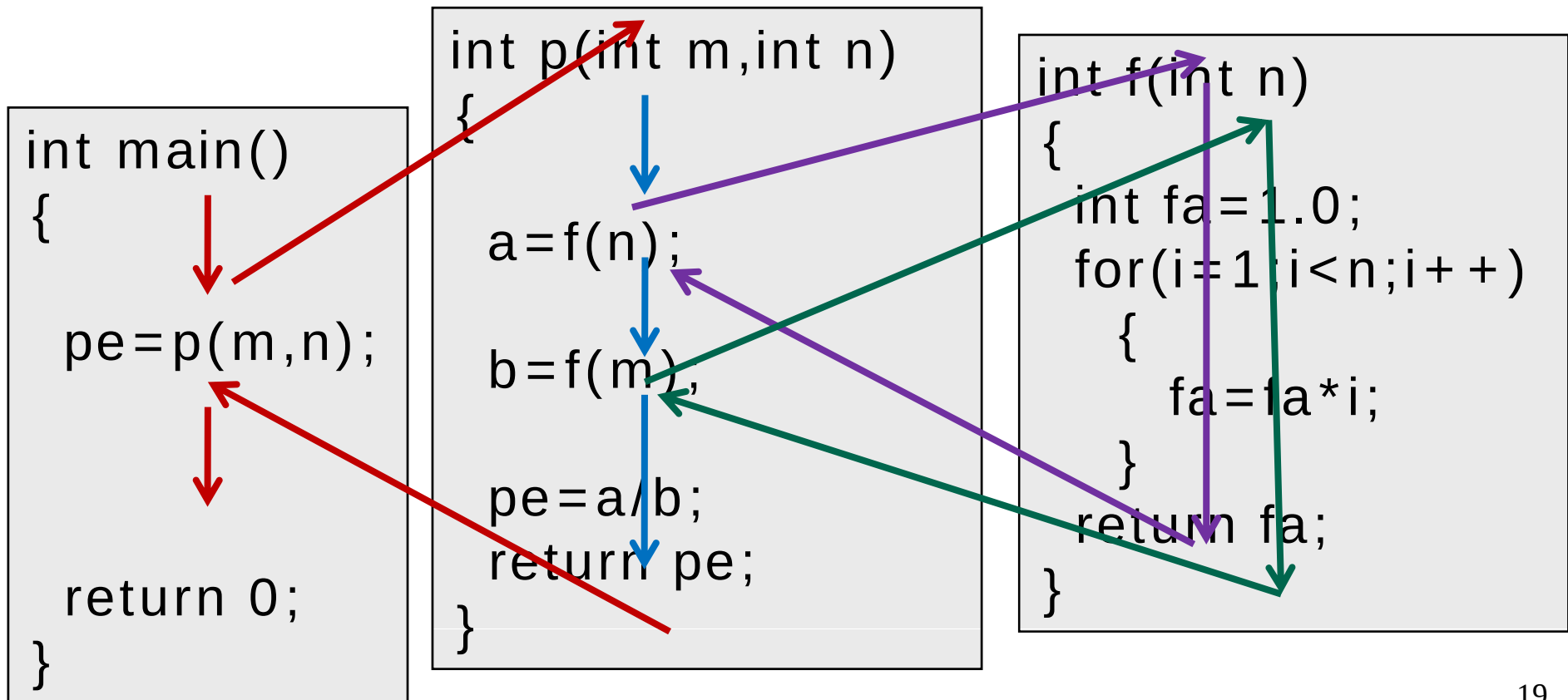
仕事を下請け、孫請けに託す。

順列の数を求めて。



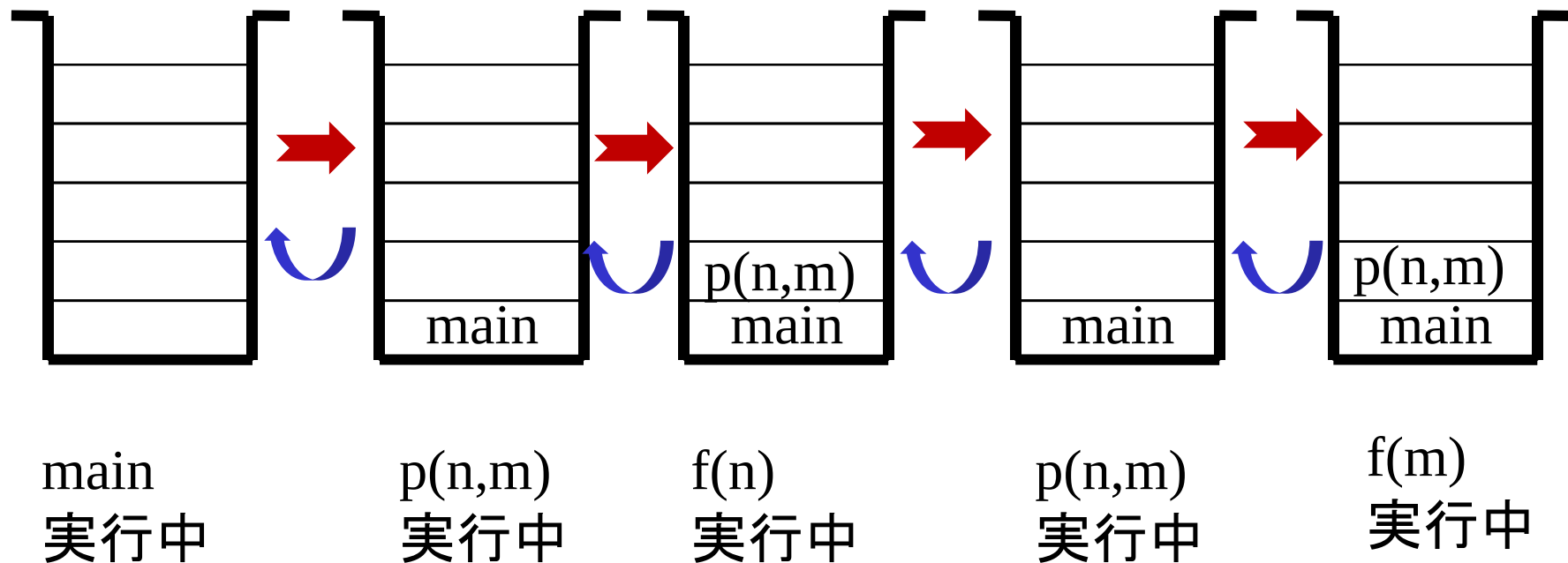
関数呼び出しと 処理の流れ

多段に呼び出された場合には、後で呼び出されたものから順に実行される。



関数呼び出しとスタック

スタックとは、後入れ先出し (Last In First Out、LIFO) のデータ構造。



➡ : 関数呼び出し

↩ : 関数からのreturn

組み合わせの数を求めるプログラム

```
/*      作成日:yyyy/mm/dd
      作成者:本荘  太郎
      学籍番号:B0zB0xx
      ソースファイル:combination.c
      実行ファイル:combination
      説明:組み合わせ数 $nCm$ を求めるプログラム。
      入力:標準入力から2つの正の整数 $n, m$ を入力。
            $n, m$ ともに15以下とする。
      出力:標準出力に組み合わせ数 $nCm$ を出力。
*/
#include <stdio.h>

/* プロトタイプ宣言 */
int    fact(int n); /* 階乗を計算する関数。*/
int    comb(int n,int m);/* 組み合わせの数 $nCm$ を計算する*/
```

```
/*    続き    */  
/* main関数 */  
int    main()  
{  
    /* ローカル変数宣言 */  
    int    n;    /* nCmのn */  
    int    m;    /* nCmのm */  
    int    com;    /* 組み合わせ数nCm */  
  
/*    次のページに続く    */
```

```

/*      続き      */
/*      入力処理 */
printf("組み合わせ数nCm を計算します。¥n");
printf("Input  n=? ");
scanf("%d",&n);
printf("Input  m=? ");
scanf("%d",&m);

/* 入力値チェック */
if(n<0||15<n||m<0||15<m||n<m)
{
    /* 不正な入力有的时候には、
       エラー表示してプログラム終了 */
    printf("不正な入力です。¥n");
    return  -1;
}

/*      正しい入力的时候、これ以降が実行される。 */
/*      次ページへ続く      */

```

```
/*      続き      */
```

```
/*      組み合わせ数を計算 */
```

```
com=comb(n,m);
```

```
/* 出力処理 */
```

```
printf("%d  C %d = %5d¥n",n,m,com);
```

```
return 0;
```

```
}
```

```
/* main関数終了 */
```

```
/*      次に行く      */
```

```

    /*      続き      */
/* 組み合わせの数を求める関数
    仮引数n:  $nCm$ のn(0以上15未満の値とする。)
    仮引数m:  $nCm$ のm (0以上15未満の値とする。)
    戻り値: 組み合わせ数 $nCm$ を返す。*/
int comb(int n,int m)
{
    /*      ローカル変数宣言      */
    int    com; /*組み合わせの数*/
    /*計算処理*/
    com=fact(n)/( fact(m)*fact(n-m) );

    return com;
}
/* 関数combの定義終 */
/* 次が続く */

```

```

        /*      続き      */
/* 階乗を求める関数
   仮引数n: n!のn(0以上15未満の値とする。)
   戻り値:n!を返す。*/
int fact(int n)
{
    /*      ローカル変数宣言      */
    int i;      /* ループカウンタ */
    int fac;    /* 階乗n! */

    fac = 1;      /* 0! = 1であるので1を代入 */

    /*      次に行く      */

```

```
/*    続き */
/* 計算処理 */
for(i=1;i<=n;i++)
{
    /* 階乗の計算 */
    fac=fac*i;
}

return fac; /* facの値n!を戻す */
}
/* 関数factの定義終 */
/* 全てのプログラム(combination.c)の終了 */
```

実行例

```
$make  
gcc combi1.c -o combi1  
$ ./combi1  
組み合わせ数nCm を計算します。  
Input n=? 4  
Input m=? 3  
4C3 = 4  
$
```

```
$/combi1  
組み合わせ数nCm を計算します。  
Input n=? 4  
Input m=? 5  
不正な入力です。  
$
```