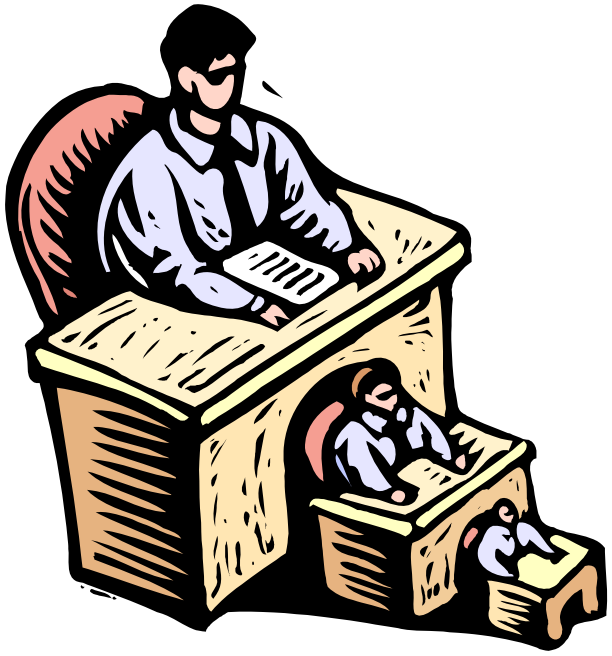
第10回　関数と再帰



今回の目標

- 再帰的な考え方に慣れる
- C言語における再帰関数を理解する。

☆階乗を求める再帰的な関数を作成し、その関数を利用するプログラムを作成する。

階乗 n!の2つの数学的表現

（1）繰り返しによる表現

$$n! = 1 \times 2 \times \cdots \times i \times \cdots \times n$$

（ $n \geq 1$ のとき。）

(なお、0!＝1）

$$= \prod_{i=1}^{n} i$$

（2）漸化式による表現

$$n! = \begin{cases} 1 & n = 0 \text{のとき} \\ n \times (n-1)! & n \geq 1 \text{のとき} \end{cases}$$

再帰

関数が自分自身を呼び出す事。

C言語では、再帰的な関数を扱える。

書式：

```
戻り値の型      関数名（仮引数の宣言付きリスト）
{
        関数名（実引数リスト）;

        return   式;
}
```

同じ関数名

注意：再帰関数は、漸化式で表わされている数学的な関数などを直接プログラミングすることができる。

再帰関数例

```
int     fact(int n)
{
        int fac;

        if(n==0)
            {
                fac=1;
            }
        else
            {
                fac=n * fact(n-1);
            }

        return fac;
}
```

再帰呼びだし
といいます。

再帰の典型的な書き方

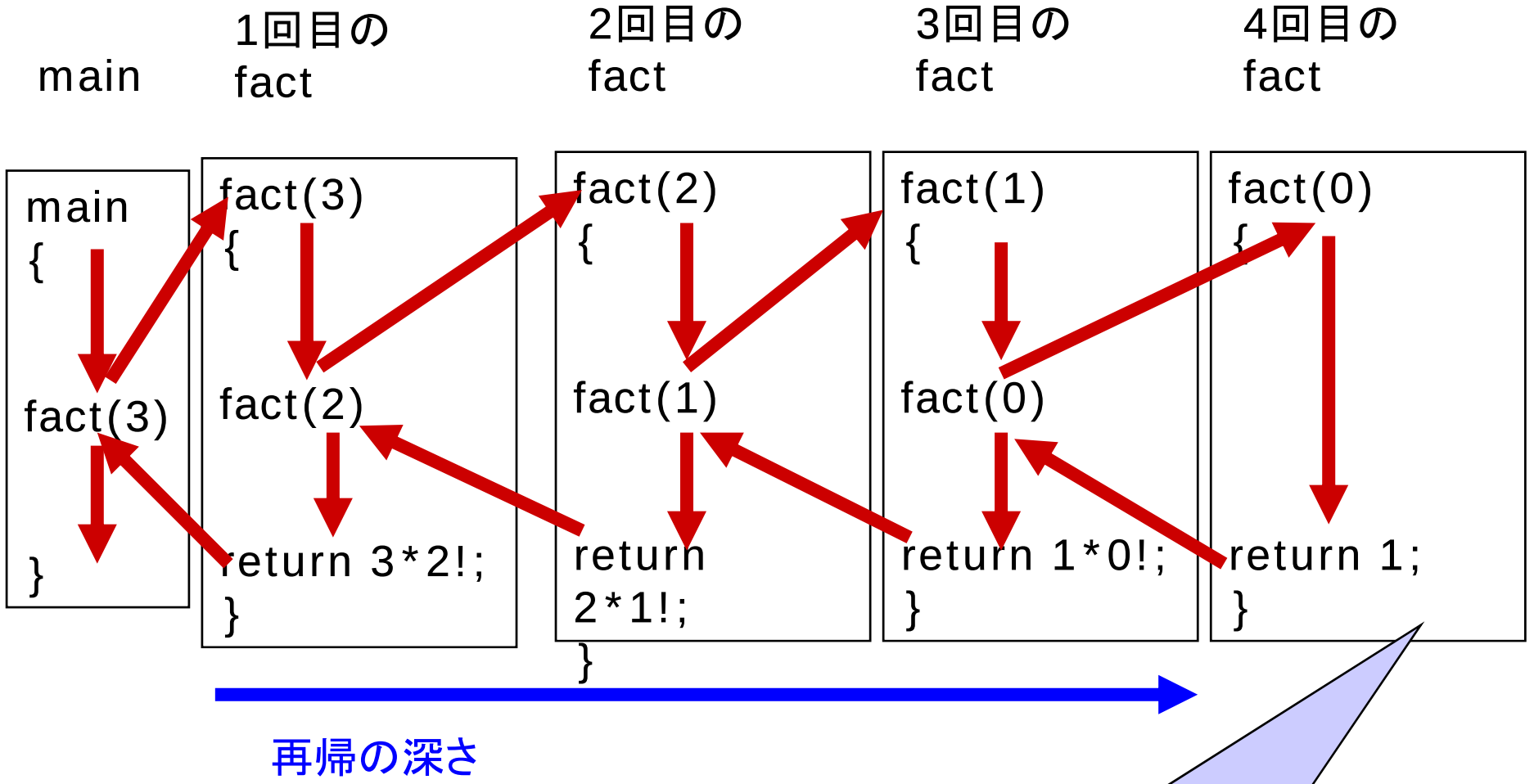
書式だけを抽出

```
int       関数名(int n)
{
        if(n==0)
          {
              /*再帰関数の基礎*/
              ans=????
          }
        else
          {
              /*再帰部分*/
              ans=関数名(n-1);
          }
        return ans;
}
```

必ず再帰の基礎（出口）をつくる事。

再帰呼び出しをするときには、
必ず出口に近づくようにすること。

プログラムの制御の流れ



再帰の基礎部分
この制御中には、
再帰呼び出しには出会わない。

再帰と変数のスコープ

たとえ同じ変数名でも、再帰の深さが違えば、
違うローカル変数として扱われる。

再帰の深さ

main

深さ1の
fact

深さ2の
fact

```
main
{
int fac;


fact(3)


}
```

```
fact(3)
{
int fac;

fact(2)



}
```

```
fact(2)
{
int fac;

fact(1)



}
```

• • •

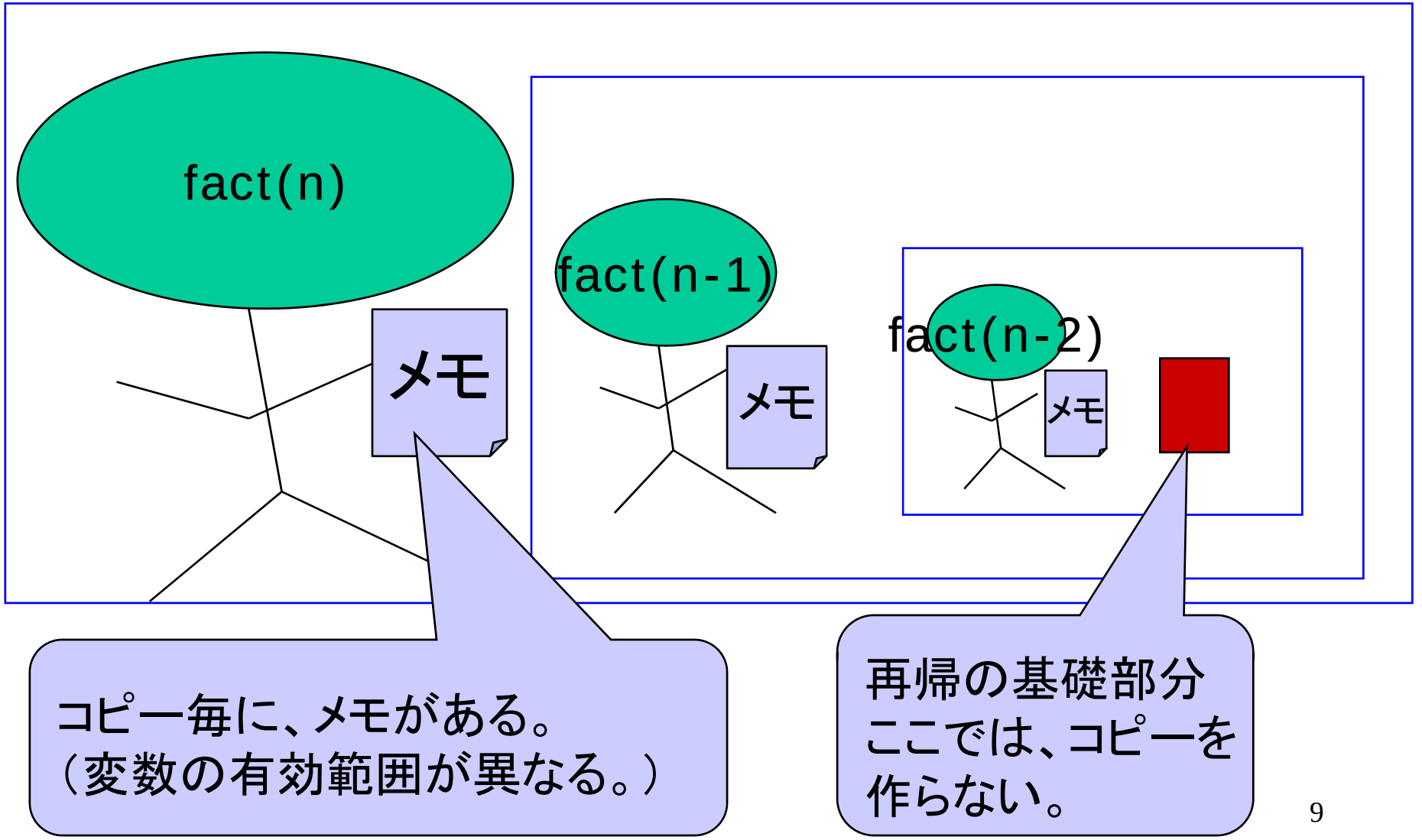
深さ0のfac

深さ1のfac

深さ2のfac

イメージ

再帰関数は、自分自身のコピーに仕事を押し付ける。



練習1

```
/* print_depth.c  再帰関数実験1     コメント等一部省略*/
#include<stdio.h>
#define  MAX   10     /*nの最大値、見栄えを制御*/
int fact(int n);       /*階乗n!を求める再帰的な関数
                           再帰の深さも表示する*/
void  print_dot(int k);   /*仮引数分ドットを表示する関数*/

int  main()
{
        int  n;          /*n! のn*/
        int fac;/*階乗*/
        printf("n= ? ");
        scanf("%d",&n);
        fac=fact(n);
        printf("¥n");
        printf("%d ! =  %5d ¥n",n,fac);
        return 0;
}/*続く*/
```

```
int fact(int  n)
{
        int  fac;

        print_dot(MAX-n);                /*再帰の深さのドット表示*/
        printf("Called by  n= %d ¥n",n);/*仮引数の値を表示*/
        if(n==0)
           {     /*再帰の基礎部分*/
                 fac=1;
                 print_dot(MAX-n);     /*再帰の深さのドット表示*/
                 printf("REACHED BASIS   0!=1  ¥n");
           }
        else
           {     /*再帰部分*/
                 fac=n*fact(n-1);
                 print_dot(MAX-n);       /*再帰の深さのドット表示*/
                 printf("%d != %d  * ( %d! ) ¥n",n,n,n-1);
           }
        return fac;
}
/*続く*/
```

```
/*仮引数分のドットを表示する関数*/
void print_dot(int k)
{
        int i;
        for(i=0;i<k;i++)
            {
                printf("..");
            }
        return;
}
/*おしまい。*/
```

終わらない再帰関数1

再帰関数は、必ず基礎（出口）部分に辿りつけないといけない。

基礎が無い再帰関数

```
/*終わらない関数1*/
int     fact(int n)
{
        int fac;

        fac=n*fact(n-1);
        return fac;
}
```

再帰関数は、ちょっと注意を怠ると、すぐに終わらないプログラムになってしまうので注意する事。

出口の無い迷路のようなもの。

プログラムが停止しないときには、
^C（コントロール＋’c’）で
停止させましょう。

終わらない再帰関数2

再帰関数は、必ず基礎（出口）部分に辿りつけないといけない。

出口に近づかない再帰関数

```
/*終わらない関数2*/
int     fact(int n)
{
        int fac;

        if(n==0)
            {
                fac=1;
            }
        else
            {
                fac=n*fact(n);
            }
        return fac;
}
```

再帰関数は、ちょっと注意を怠ると、すぐに終わらないプログラムになってしまうので注意する事。

無限増殖関数の出来上がり。

プログラムが停止しないときには、
^C（コントロール＋’c’）で
停止させましょう。

終わらない再帰関数3

再帰関数は、必ず基礎（出口）部分に辿りつけないといけない。

出口から遠ざかる再帰関数

```
/*終わらない関数3*/
int     fact(int n)
{
        int fac;

        if(n==0)
            {
                fac=1;
            }
        else
            {
                fac=n*fact(n+1);
            }
        return fac;
}
```

再帰関数は、ちょっと注意を怠ると、すぐに終わらないプログラムになってしまうので注意する事。

無限増殖関数が出来上がる。

プログラムが停止しないときには、
^C（コントロール＋’c’）で
停止させる。

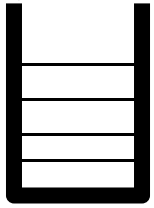
再帰と繰り返し

再帰的な関数は、繰り返しを用いて書き直せることがある。
再帰を用いてプログラミングするか、
繰り返しを用いてプログラミングするかは、
よく考えて決めること。

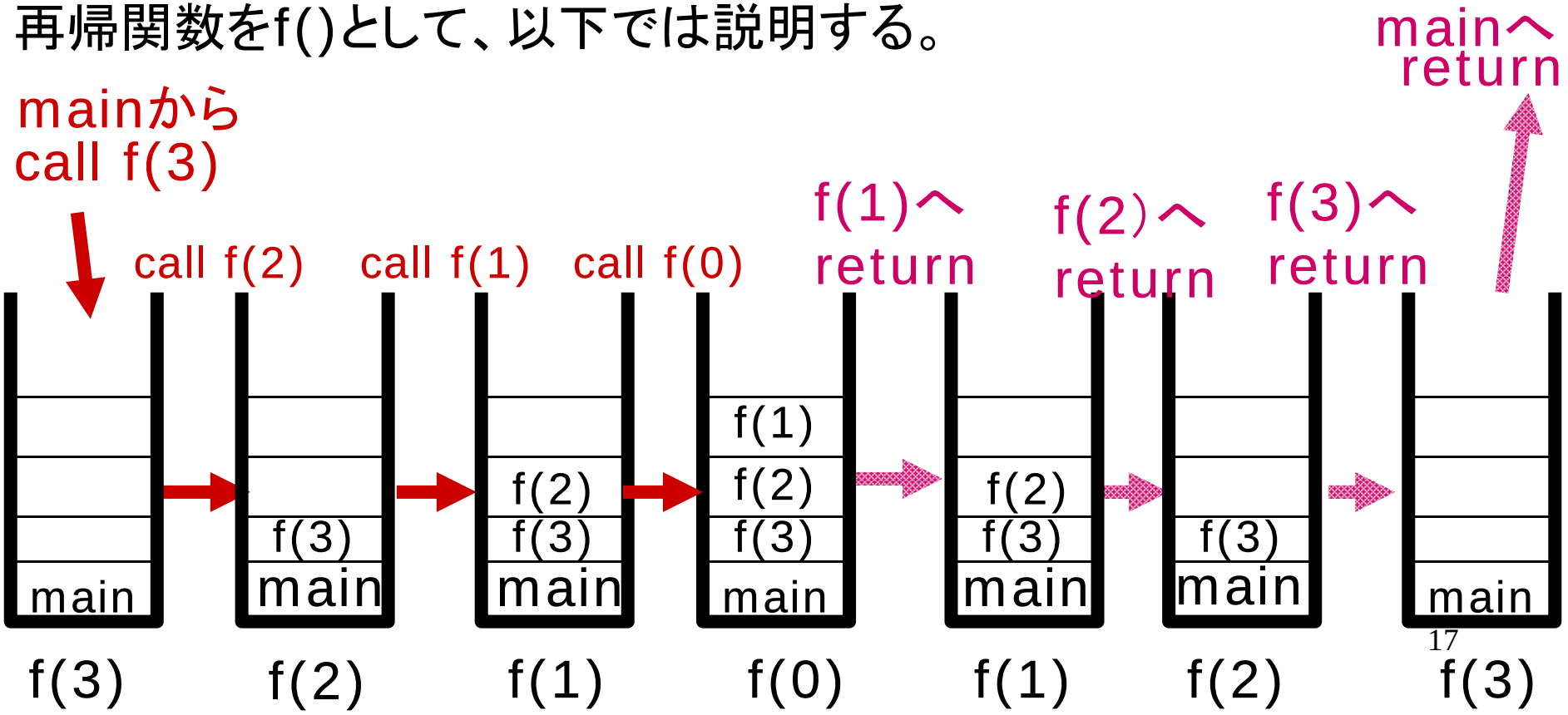
```
int  fact(int n)
{
        int     i;
        int fac;
        fac=1;
        for(i=1;i<=n;i++)
            {
                fac =fac*i;
            }
        return fac;
}
```

再帰とスタック

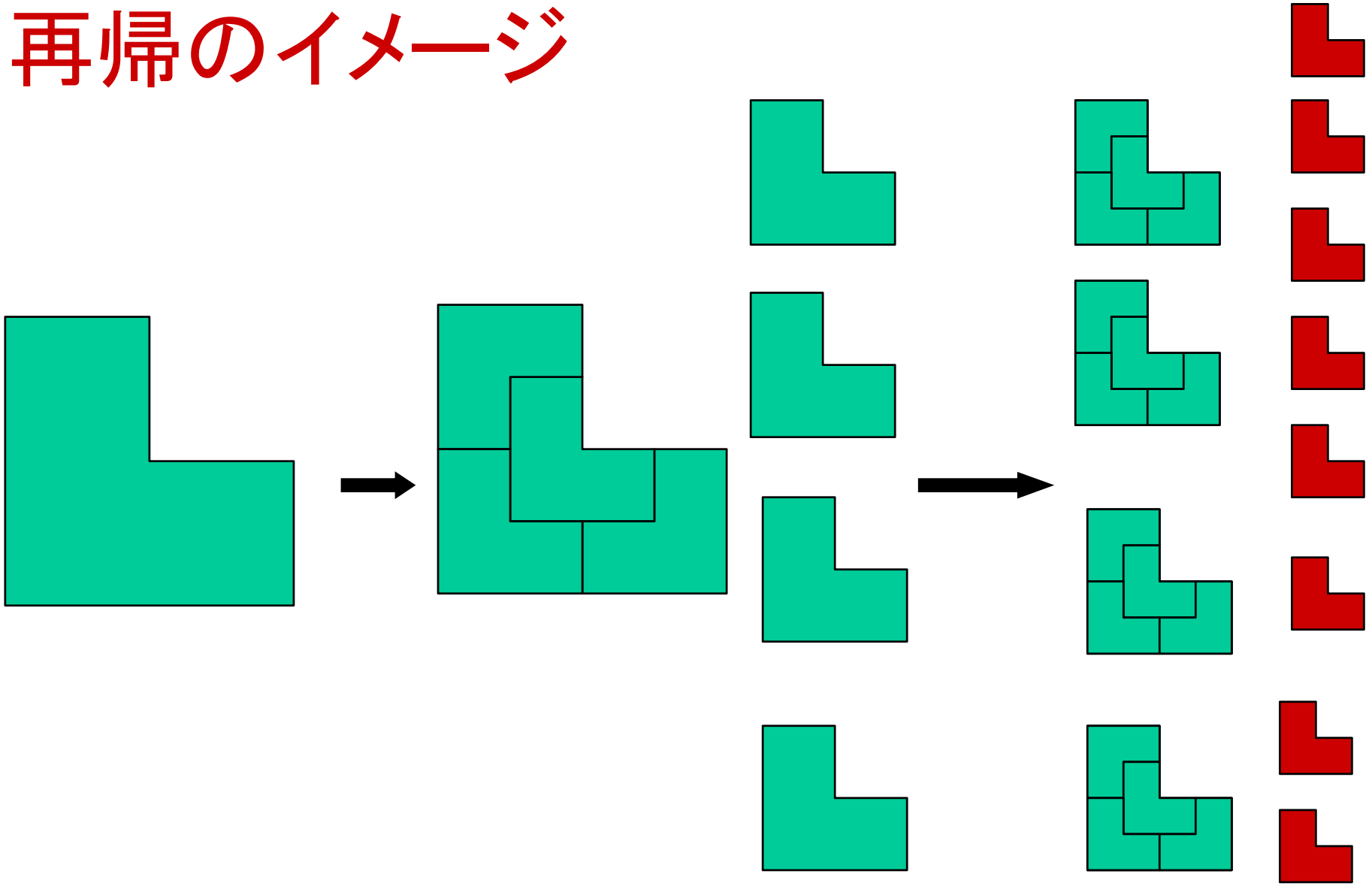
スタックとは、後入れ先出し(Last In First Out,LIFO)のデータ構造



再帰関数はその呼ばれた順序にスタックにつまれ、
最後に呼ばれたものから終了する。
再帰関数をf()として、以下では説明する。



再帰のイメージ

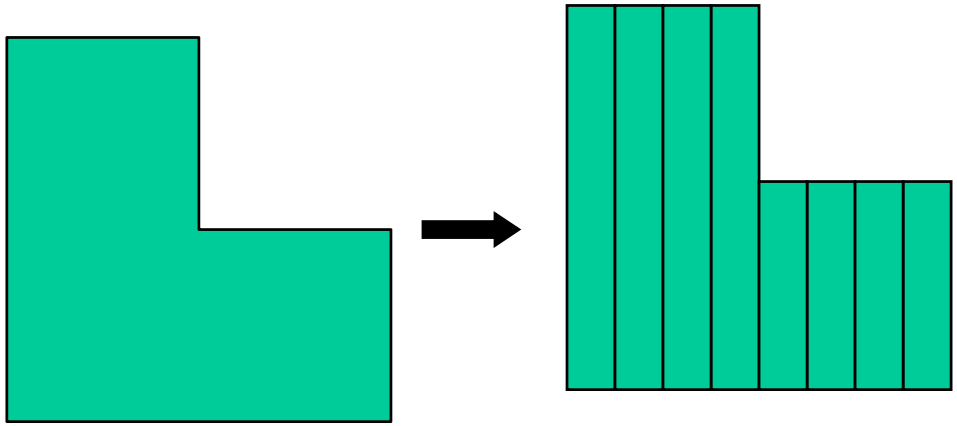


元の問題

”同型”の
小問題に分割

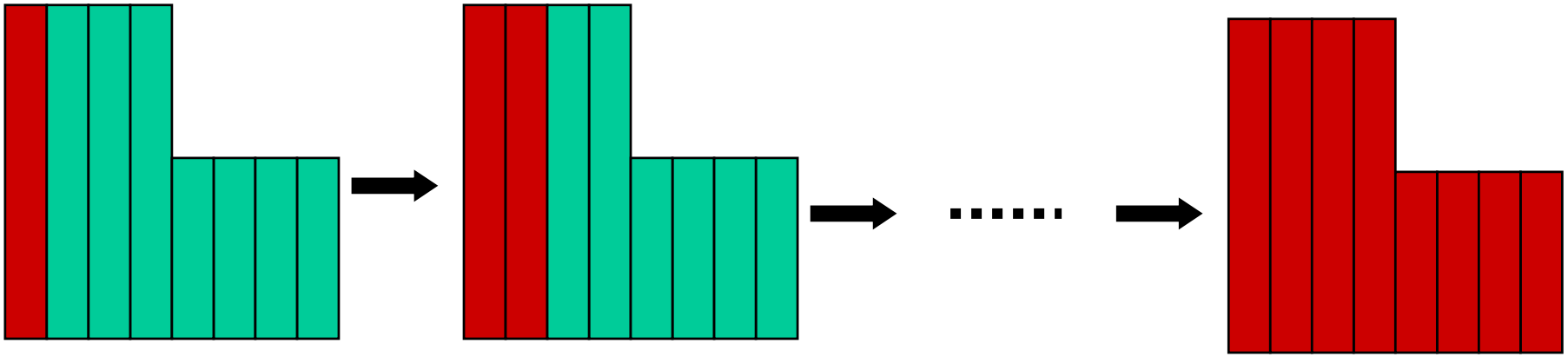
小さい問題は
各個撃破

繰り返しのイメージ



元の問題

”均等”に
小さく分割



小さい問題は各個撃破

再帰的に階乗を求めるプログラム（p.123)

```
/*
        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：fact_rec.c
        実行ファイル：fact_rec
        説明：再帰的にn!を求める関数を用いて、
                n!を計算する。
        入力：標準入力から0から15までの整数nを入力。
        出力：標準出力にn!を出力。
*/
#include<stdio.h>
/*プロトタイプ宣言*/
int   fact(int n);          /*階乗n!を求める再帰的な関数*/
/*      次のページに続く      */
```

```
int     main()
{       /*ローカル変数宣言*/
        int     n;      /*n!のn*/
        int     fac;    /*階乗n!*/
        /*入力処理 */
        printf("階乗n!を計算します。¥n");
        printf("n=? ¥n");
        scanf("%d",&n);

        /* 入力値チェック */
        if(n<0||15<n)
            {
                printf(“nは0から15までの整数にする。¥n");
                return  -1;
            }
        /*これ以降では0から15までの整数*/
/*続く*/
```

```
        /*      続き    */

        /*階乗n!の計算 */
        fac=fact(n);

        /*出力処理*/
        printf("%d ! = %10d ¥n",n,fac);

        return 0;
}

/*続く */
```

```
/*階乗を求める再帰的関数
        仮引数n: n!のn,(nは0から15)
        戻り値:n! */
int     fact(int n)
{       /*ローカル変数宣言*/
        int fac;        /*階乗n!*/

        /*次に続く*/
```

```
        /*漸化式に基づく、階乗の定義式*/
        if(n==0)
            {
                /*再帰の基礎、0!=1*/
                fac=1;
            }
        else
            {
                /*再帰部分、n!=n*(n-1)!*/
                fac=n * fact(n-1);/*再帰呼び出し*/
            }
        return fac;
}
```

実行例

```
$make
gcc  fact_rec.c  -o       fact_rec
$ ./fact_rec
階乗n!を計算します。
 n=?
5
5! =          120
$
```

```
$ ./fact_rec
階乗n!を計算します。
 n=?
-1
nは0から15までの整数にする。
$
```