

# 2021 計算機プログラミング演習

第13回  
ユーザ定義関数(2)

# 第13回の内容

- 再帰呼び出し(Recursive call)を用いた処理
- ユーザー定義関数やこれまでの講義で扱った内容をもとにプログラムを作成します

# 再帰呼び出し(Recursive call)

- ある関数(自分で定義した関数)の中で, その関数を呼び出すことを再帰呼び出しという
- forやwhileと同様に繰り返し処理に用いる
- 但し, 終了条件を記述しないと無限ループになる

```
int function( ..... ){           //関数functionの定義  
  
.....  
function ( );    //再帰呼び出し  
.....  
  
}
```

# 再帰呼び出しの例(階乗)

```
1  #define _CRT_SECURE_NO_WARNINGS 1
2  #include <stdio.h>
3
4
5  int factorial(int n); //プロトタイプ宣言
6
7
8  int main(void) {
9
10     int num1 = 0;
11     int num2 = 0;
12
13     printf("正の整数の階乗を計算します\n");
14
15     scanf("%d", &num1);
16
17     printf("%dの階乗の計算を行います\n", num1);
18
19     num2 = factorial(num1);
20
21     printf("%d\n", num2);
22
23
24     return 0;
25 }
```

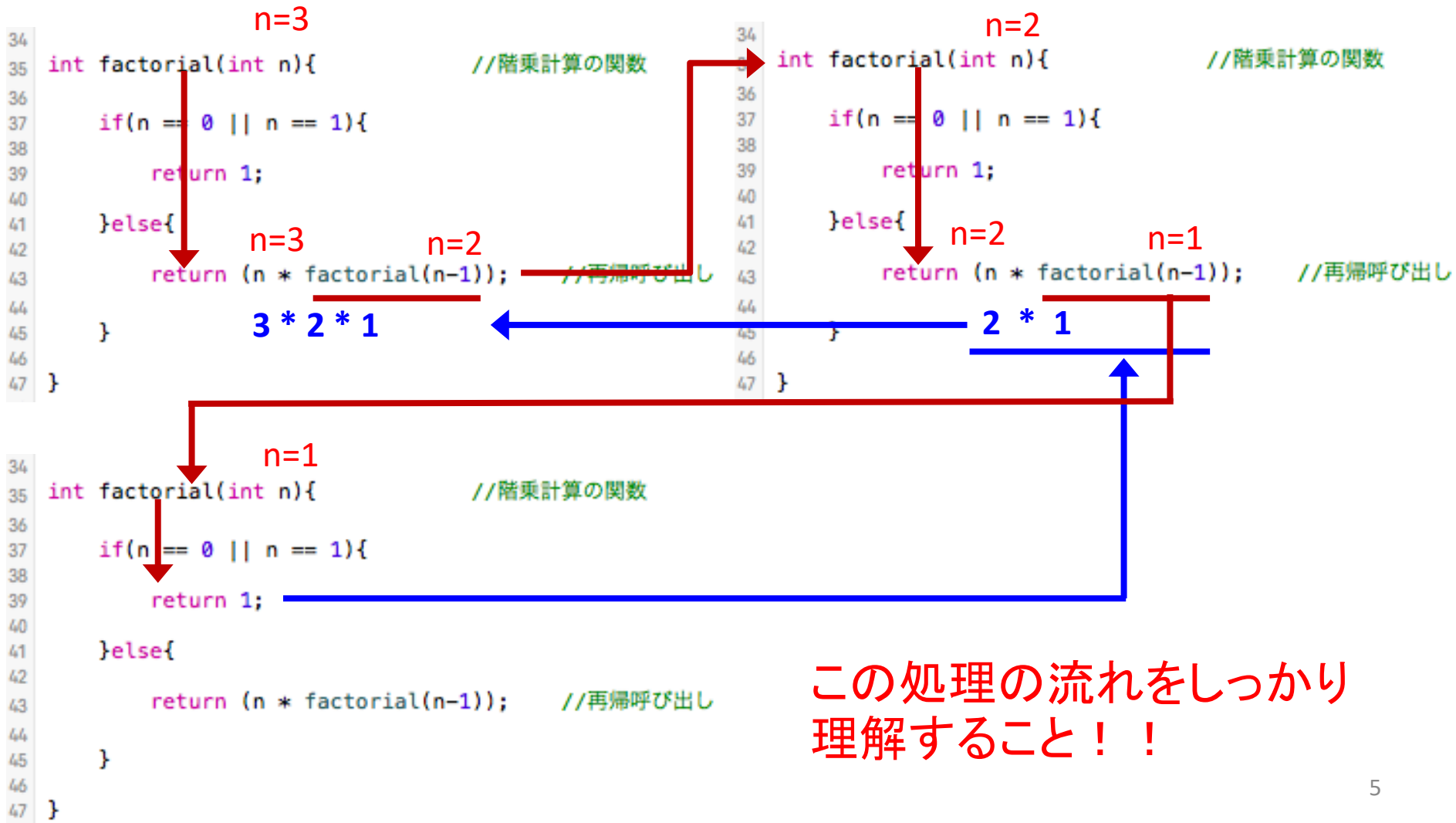
再帰呼び出しを用いた階乗の計算  
プログラムである

```
C:\Users\Nakano\source\repos\Kaijyo_saiki\Debug>Kaijyo_saiki
正の整数の階乗を計算します
8
8の階乗の計算を行います
40320
```

```
27
28 int factorial(int n) { //階乗計算の関数
29
30     if (n == 0 || n == 1) {
31
32         return 1;
33     }
34
35     else {
36
37         return (n * factorial(n - 1)); //再帰呼び出し
38
39     }
40
41 }
42
```

# 再帰呼び出しの例(階乗)

再帰呼び出しによる処理の流れ(n=3の場合)



# 再帰呼び出しの留意事項

- 再帰呼び出しの回数にはメモリ容量により制限がある
- 再帰呼び出しの回数が多いと与えられたメモリの領域(スタック)を使い果たしてしまう
- 実際には無限ループになることなくエラーが発生する
- 再帰呼び出しを使うとより計算式に近いより簡潔な記述ができることが多い(例えば 数列, 漸化式など)

# 演習1 ベクトル行列演算

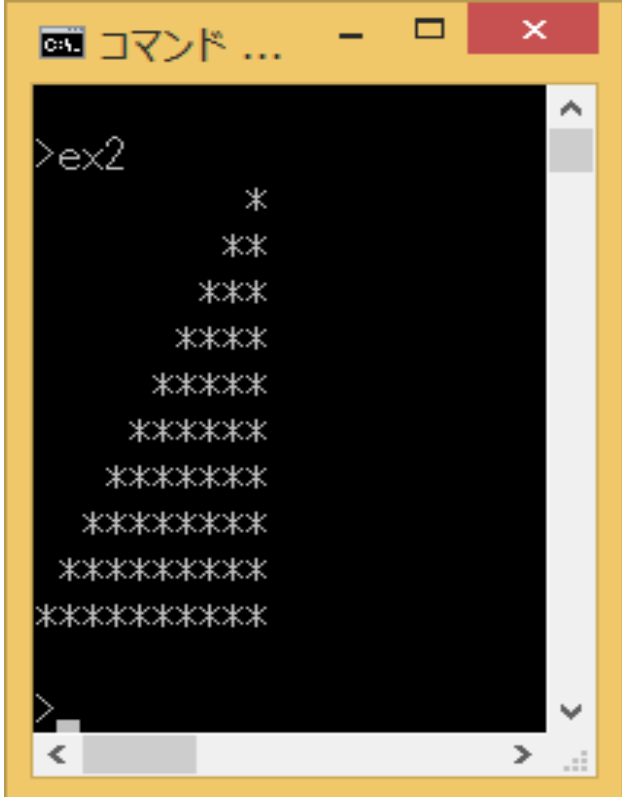
3行3列の行列  $A$  と列ベクトル  $x$  の掛け算を行うプログラムを作成せよ

$$b = Ax = \begin{bmatrix} 1.2 & 2.3 & 3.4 \\ 5.8 & 2.1 & -4.9 \\ 3.2 & 6.4 & 5.2 \end{bmatrix} \begin{bmatrix} 3.5 \\ 2.1 \\ 8.8 \end{bmatrix}$$

## 演習2 for文の復習

図に示すように10段の直角三角形（右下が直角）を表示するプログラムを作成せよ

- 配列を使わないこと
- 二重ループを使うこと



```
C:\> コマンド ...
>ex2
      *
     **
    ***
   ****
  *****
 *****
*****
*****
*****
*****
```



# 演習3 再帰呼び出し(フィボナッチ数列)

$$a_0 = 0, a_1 = 1, a_k = a_{k-2} + a_{k-1} \quad (k = 2, 3, \dots)$$

フィボナッチ数列を求めるプログラムを再帰呼び出しを用いて作成し,  $k = 10$ のときの  $a_{10}$ の値を求めよ

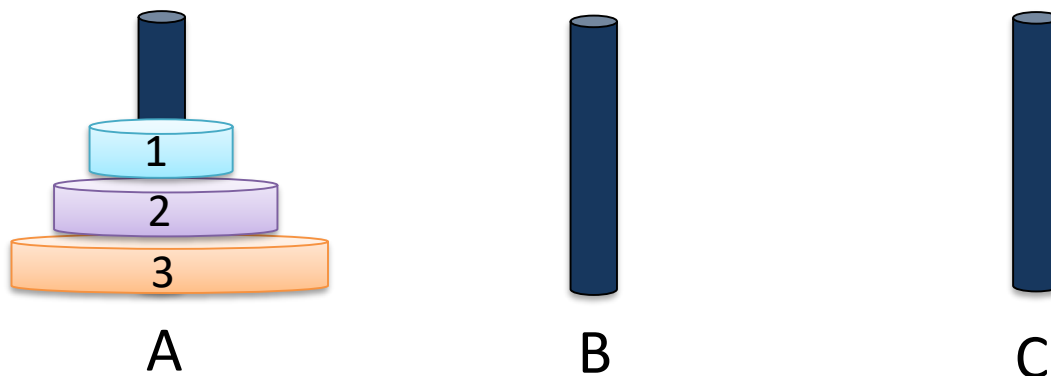
- プロトタイプ宣言を忘れずに記述すること
- 無限ループにならないように注意すること
- forやwhileを使わないこと

```
C:\Users\Nakano\source\repos\fibonacci\Debug>fibonacci
何番目のフィボナッチ数列の値を求めたいか？
10
プログラム実行結果  a10 = 55
```

# 演習4 再帰呼び出し(ハノイ塔)

## ハノイの塔

フランスのE.Lucasによって作られたゲーム(1883年ころ)

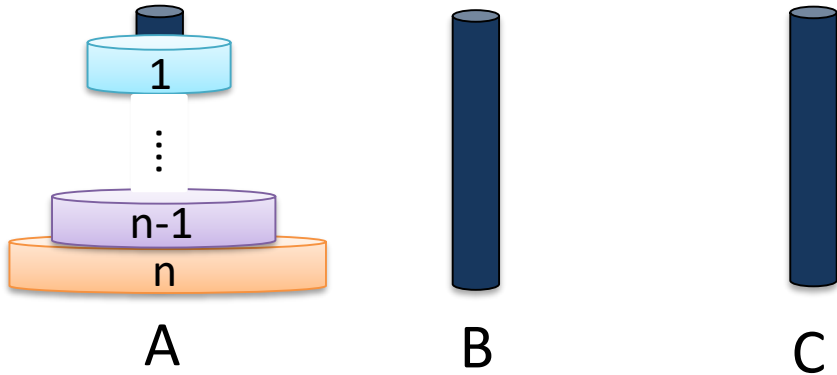


水平な地面に3本の棒を立て、それぞれ A, B, C と名前を付けている. また、この棒を通る穴が空いたサイズの異なる番号の付いた円盤が3枚ある. この3枚の円盤を AからCに移動させたい. ただし以下のことが条件である

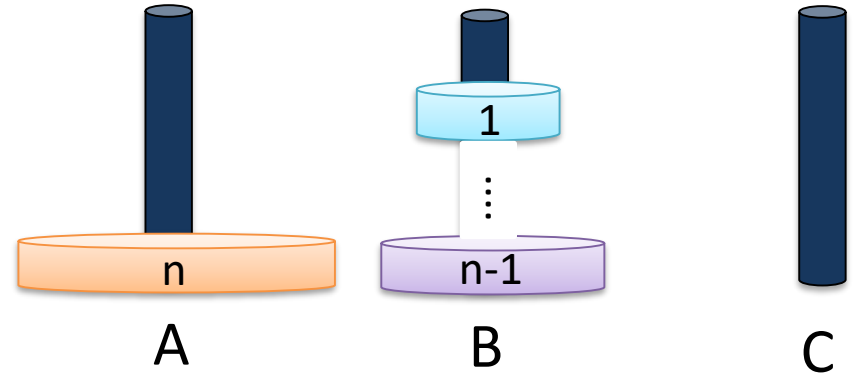
- 1回の移動で移動できる円盤の枚数は1枚だけ
- 小さい円盤の上に大きい円盤を載せることはできない
- 必ず円盤は棒A,B,Cのどれかに置かなければならない

# 演習3 再帰呼び出し(ハノイ塔)

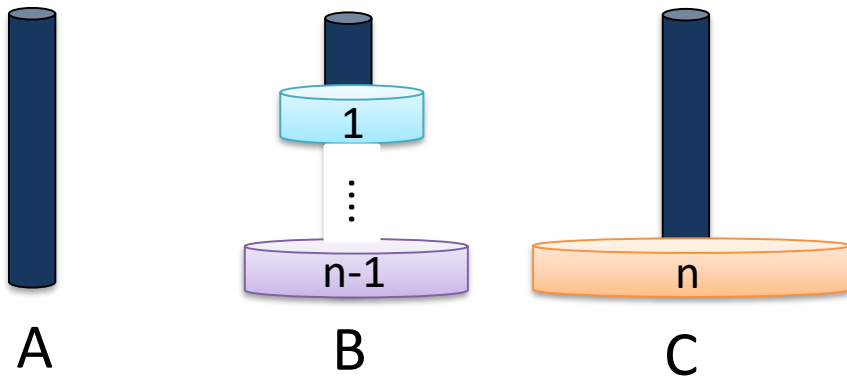
① 移動スタート



② 1からn-1の円盤をBに移動



③ nの円盤をCに移動

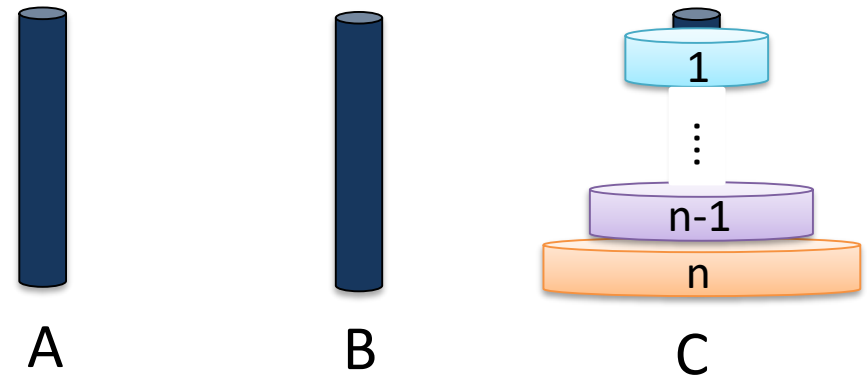


(from)

(work)

(to)

④ 1からn-1の円盤をCに移動



(from)

(work)

(to)

# ハノイの塔のプログラム

- 作成する関数の中で再帰呼び出しを記述すること
- 再帰を用いるということは、何らかの繰り返しの規則性が存在する
- プログラムを実行して表示するのは円盤の移動の履歴である

```
C:\Users\Nakano>cd C:\Users\Nakano\source\repos\Hanoi\Debug
C:\Users\Nakano\source\repos\Hanoi\Debug>Hanoi
1番目の円盤をAからCへ移動
2番目の円盤をAからBへ移動
1番目の円盤をCからBへ移動
3番目の円盤をAからCへ移動
1番目の円盤をBからAへ移動
2番目の円盤をBからCへ移動
1番目の円盤をAからCへ移動
```

# 演習4のプログラム例

```
1
2
3     #include <stdio.h>
4
5
6     void hanoi(int n, char* from, char* work, char* dest);
7
8     int main(void) {
9
10         hanoi(3, "A", "B", "C");
11
12         return 0;
13     }
14
15     void hanoi(int n, char* from, char* work, char* to)
16     {
17         if (n > 0) {
18             [REDACTED]
19
20             printf("%d番目の円盤を%sから%sへ移動\n", n, from, to);
21
22             [REDACTED]
23
24         }
25     }
26
27
```

# 演習1のプログラム例

```
#include <stdio.h>
int main(void) {
    double a[3][3]={ {1.2, 2.3, 3.4}, {5.8,2.1,-4.9},
                     {3.2,6.4,5.2} };
    double x[3]={3.5, 2.1, 8.8};
    double b[3];
    int i, j ;

    for ( i = 0 ; i < 3 ; i ++ ) {
        b[i] = 0.0;
        for ( j = 0 ; j < 3 ; j++ ) {
            b[i]+= a[i][j]*x[j];
        }
        printf(" b [ %d ] = %f\\n",
               i, b[ i ] );
    }
    return 0;
}
```

# 演習2のプログラム例

```
#include <stdio.h>

int main(void) {

    int i, j;

    for (i = 1; i <= 10; i++) {
        for (j = 1; j <= 10 - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= i; j++) {
            printf("*");
        }
        printf("¥n");
    }
    return 0;
}
```

# 演習3のプログラム例

```
1  #define _CRT_SECURE_NO_WARNINGS 1
2  #include <stdio.h>
3
4  int fib(int k);
5
6  int main(int argc, const char* argv[]) {
7
8      int max = 0;
9      printf("何番目のフィボナッチ数列の値を求めたいか？\n");
10     scanf("%d", &max);
11
12     printf("プログラム実行結果  a10 = ");
13     printf("%d\n", fib(max));
14
15
16     return 0;
17 }
18
19
20 int fib(int k) {
21
22     int n = 0;
23
24     if (k == 0) {
25
26         return 0;
27     }
28
29     if (k == 1) {
30
31         return 1;
32     }
33
34
35     if (k > 1) {
36
37         n = fib(k - 1) + fib(k - 2);
38     }
39
40     return n;
41 }
42
43
```

```
C:\Users\Nakano\source\repos\fibonacci\Debug>fibonacci
何番目のフィボナッチ数列の値を求めたいか？
10
プログラム実行結果  a10 = 55
```



# 演習4のプログラム例

```
1
2
3  #include <stdio.h>
4
5
6  void hanoi(int n, char* from, char* work, char* dest);
7
8  int main(void) {
9      hanoi(3, "A", "B", "C");
10
11      return 0;
12  }
13
14
15  void hanoi(int n, char* from, char* work, char* to)
16  {
17      if (n > 0) {
18          hanoi(n - 1, from, to, work);
19          printf("%d番目の円盤を%sから%sへ移動\n", n, from, to);
20          hanoi(n - 1, work, from, to);
21      }
22  }
```

# 演習4の再帰呼び出し部分

```
14
15 void hanoi(int n, char* from, char* work, char* to)
16 {
17     if (n > 0) {
18         hanoi(n - 1, from, to, work);
19         printf("%d番目の円盤を%sから%sへ移動\n", n, from, to);
20         hanoi(n - 1, work, from, to);
21     }
22 }
23
24
25
26
27
```

3 A B C

2 A C B

2 A C B

1 A B C

1 A B C

1 A B C

0 A C B

1 A C

```
C:\Users\Nakano>cd C:\Users\Nakano\source\repos\Hanoi\Hanoi
C:\Users\Nakano\source\repos\Hanoi\Hanoi>
1番目の円盤をAからCへ移動
2番目の円盤をAからBへ移動
1番目の円盤をCからBへ移動
3番目の円盤をAからCへ移動
1番目の円盤をBからAへ移動
2番目の円盤をBからCへ移動
1番目の円盤をAからCへ移動
```

残りの再帰呼び出し  
は自分で確認すること