cp-10. 末尾再帰関数と多重再帰関数

（C プログラミング入門）

URL: https://www.kkaneko.jp/pro/adp/index.html

金子邦彦

内容

例題１．　フィボナッチ数列

例題２．　McCarthyの91関数

例題３．　Ackermann関数

例題４．　総和を求める末尾再帰関数

目標

- ２個以上の再帰呼出しを含む再帰関数（多重再帰関数）のプログラムを読み，再帰関数について理解を深める

- 繰り返し文を使ったプログラムを，末尾再帰関数の形に書き直すことができるようになる

２個以上の再帰呼出しを含む多重再帰関数

関数Fの本体に，Fの再帰呼出しを２個以上書いてもよい.

例）

- ハノイの塔
- フィボナッチ数列
- McCarthyの91関数
- Ackermann関数

例題１．フィボナッチ数列

- フィボナッチ数列

$$f_0 = 1,$$

$$f_1 = 1,$$

$$f_n = f_{n-1} + f_{n-2} (n > 1)$$

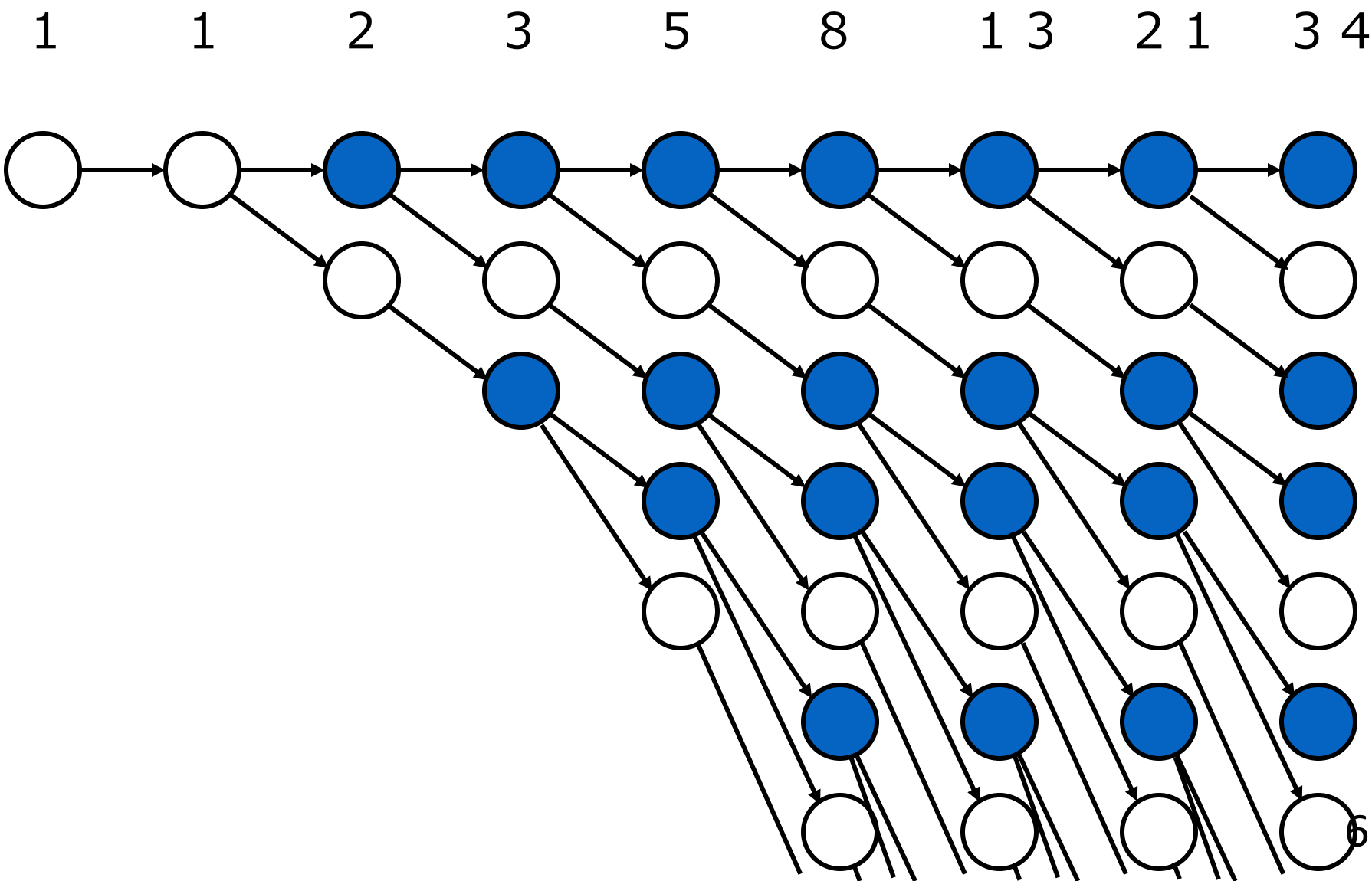
の i 番めの数　を計算するプログラムを，

再帰関数を用いて作る

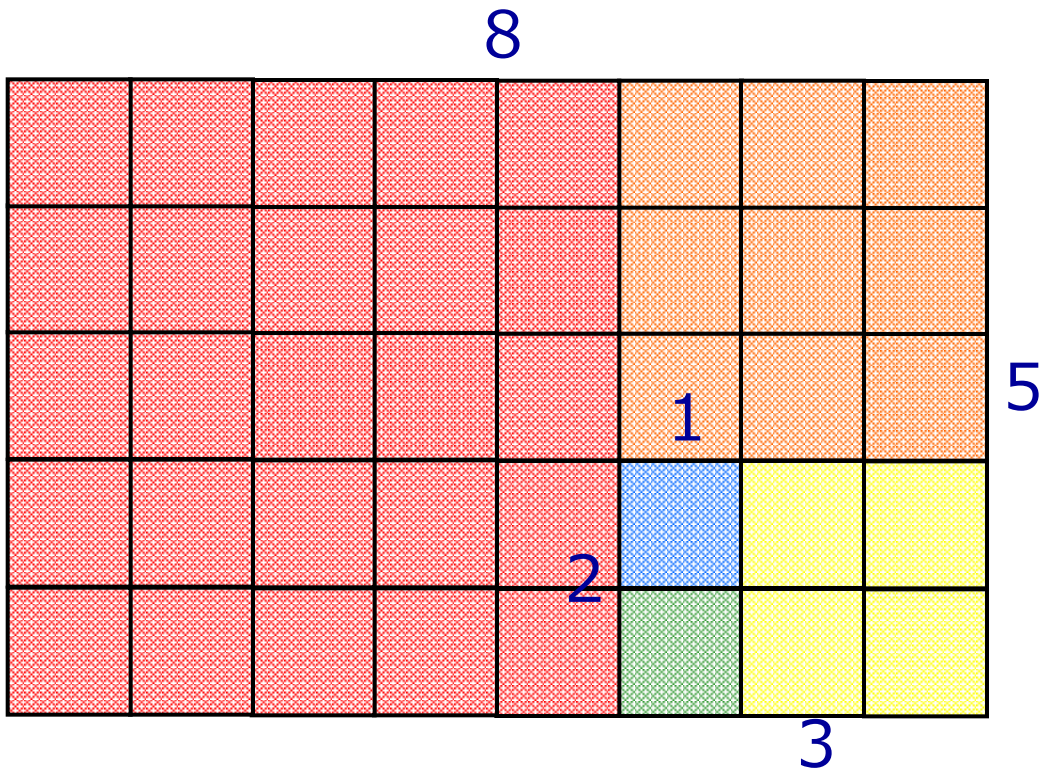
例）　1,1,2,3,5,8,13,21,34,55,89,144,….

f_i

フィボナッチ数列



フィボナッチ数列



```
#include <stdio.h>
#pragma warning(disable:4996)
int main()
{
    int i;
    int n;
    int fn;
    int fn1;
    int fn2;
    printf("n=");
    scanf("%d", &n);
    fn1 = 1;
    fn2 = 1;
    for (i=2; i<=n; i++) {
        fn=fn1+fn2;
        fn2=fn1;
        fn1=fn;
        printf("f(%d) = %d¥n", i, fn);
    }
    return 0;
}
```

条件式

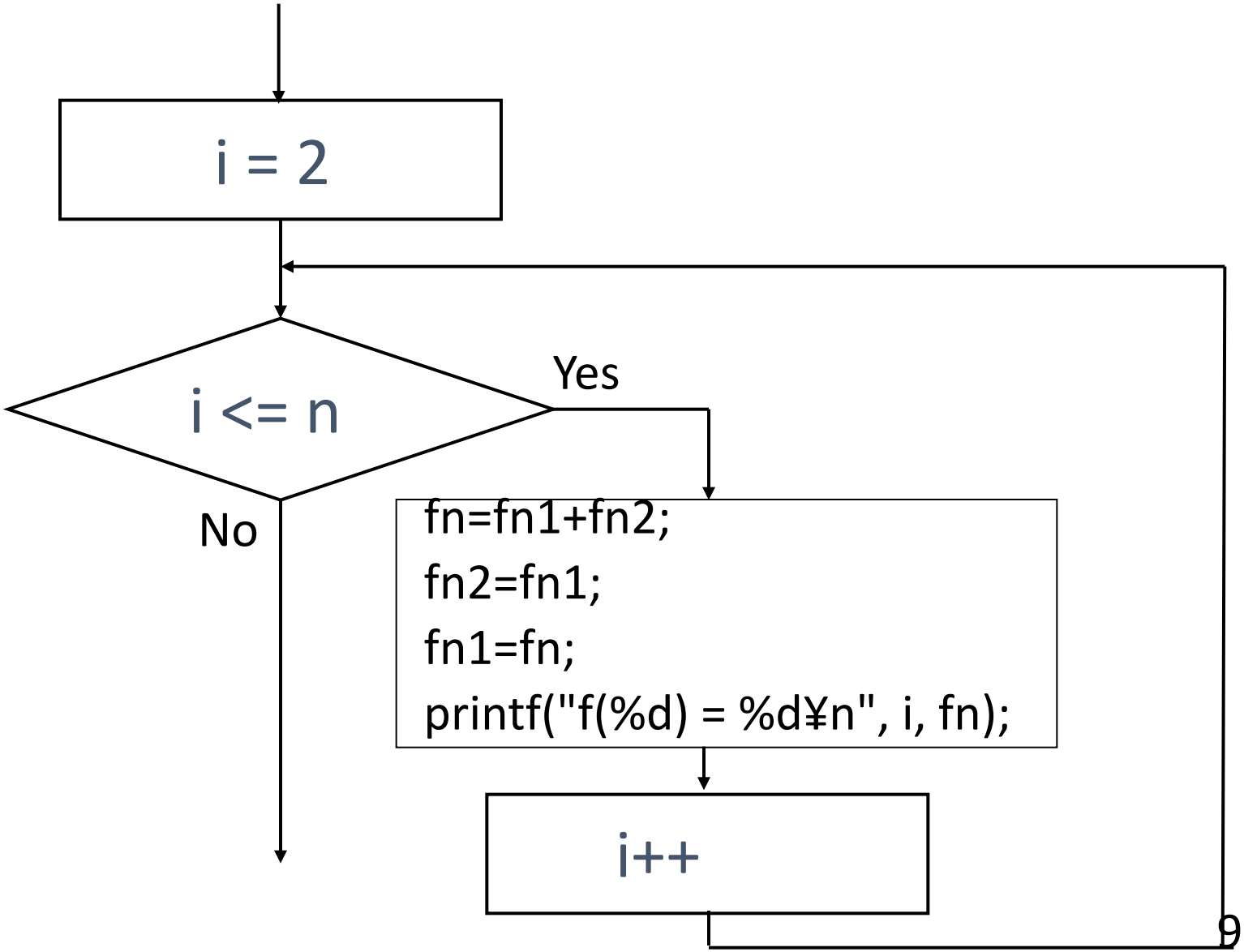
条件が成り立つ限り，
実行されつづける部分

順番に意味がある．

```
fn1=fn;
fn2=fn1;
```

とはしないこと

「繰り返し」によるフィボナッチ数列



「繰り返し」によるフィボナッチ数列

n = 5 とすると

i = 2	i <= 5 が成立する　fn = 1 + 1; fn2 = 1; fn1 = 2
i = 3	i <= 5 が成立する　fn = 2 + 1; fn2 = 2; fn1 = 3
i = 4	i <= 5 が成立する　fn = 3 + 2; fn2 = 3; fn1 = 5
i = 5	i <= 5 が成立する　fn = 5 + 3; fn2 = 5; fn1 = 8
i = 6	i <= 5 が成立しない

fi が入る　fi-1 が入る　fi が入る

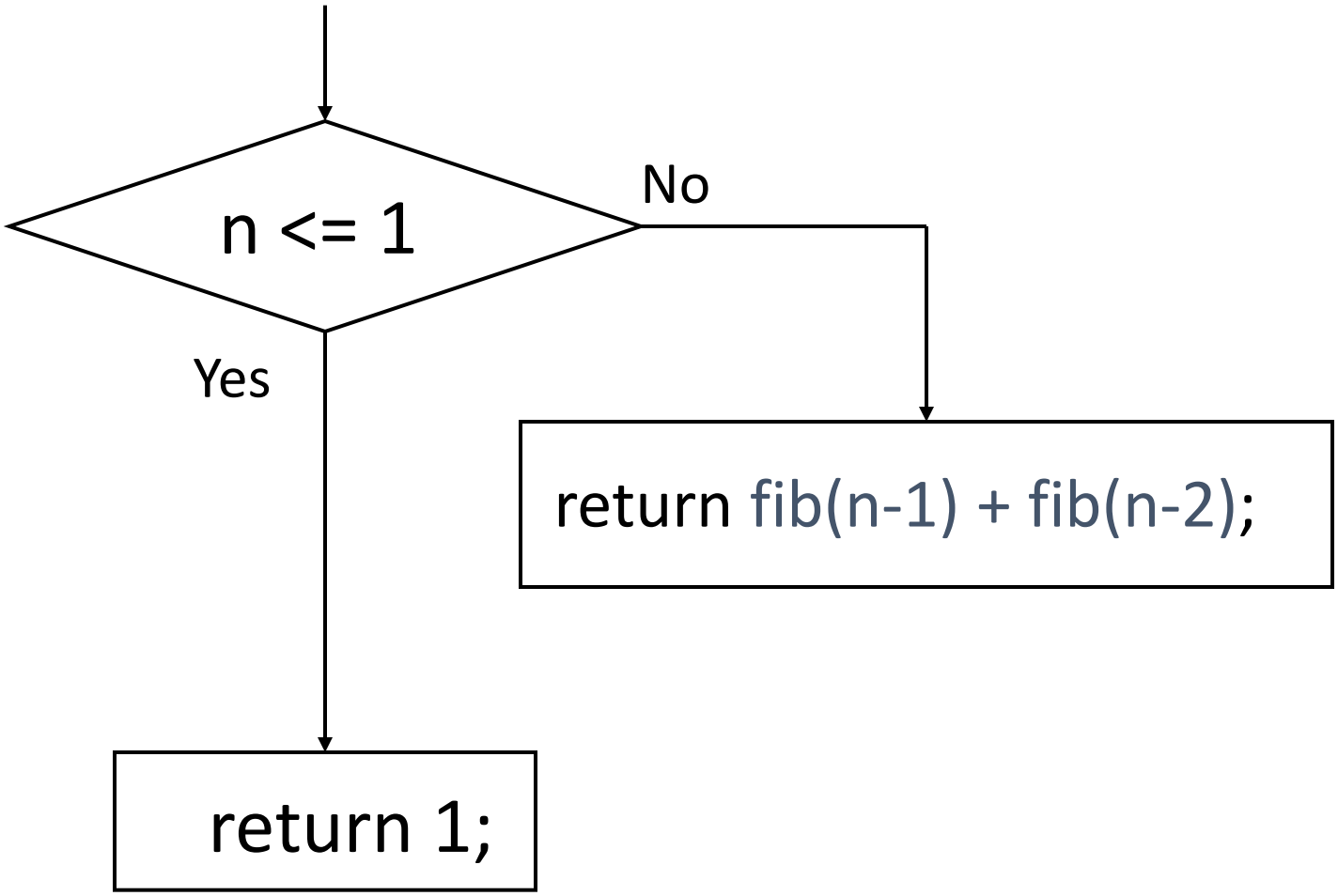
```
#include <stdio.h>
#pragma warning(disable:4996)
int fib(int n)
{
  if (n<=1) {
    return 1;
  }
  else {
    return fib(n-1)+fib(n-2);
  }
}
int main()
{
  int n;
  int fn;
  printf("Enter a number: ");
  scanf("%d",&n);
  fn = fib(n);
  printf("Fib(%d)=%d¥n",n ,fn);
  return 0;
}
```

フィボナッチ数列

実行結果の例

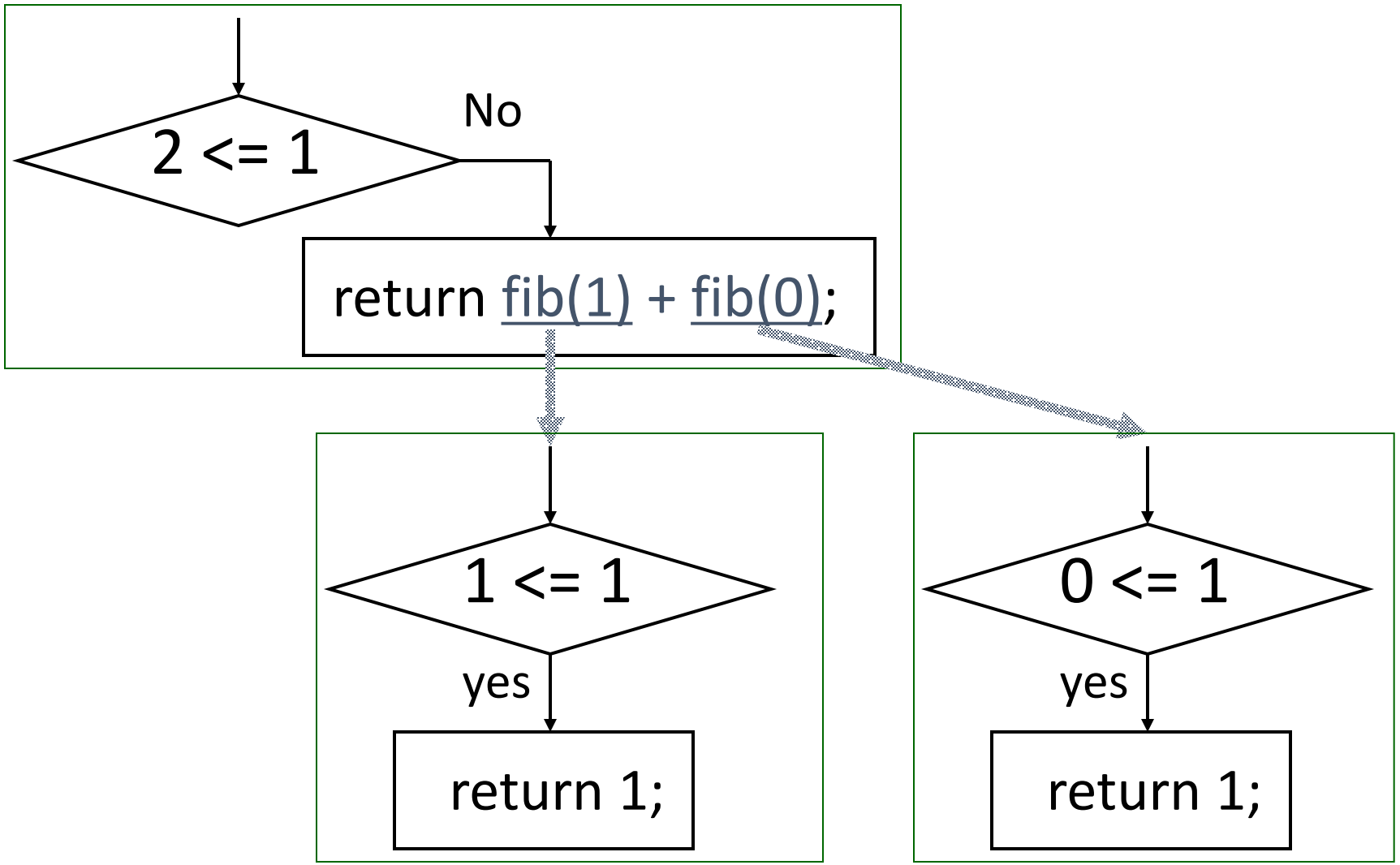
```
Enter a number: 5

Fib(5)=8
```

「再帰」によるフィボナッチ数列



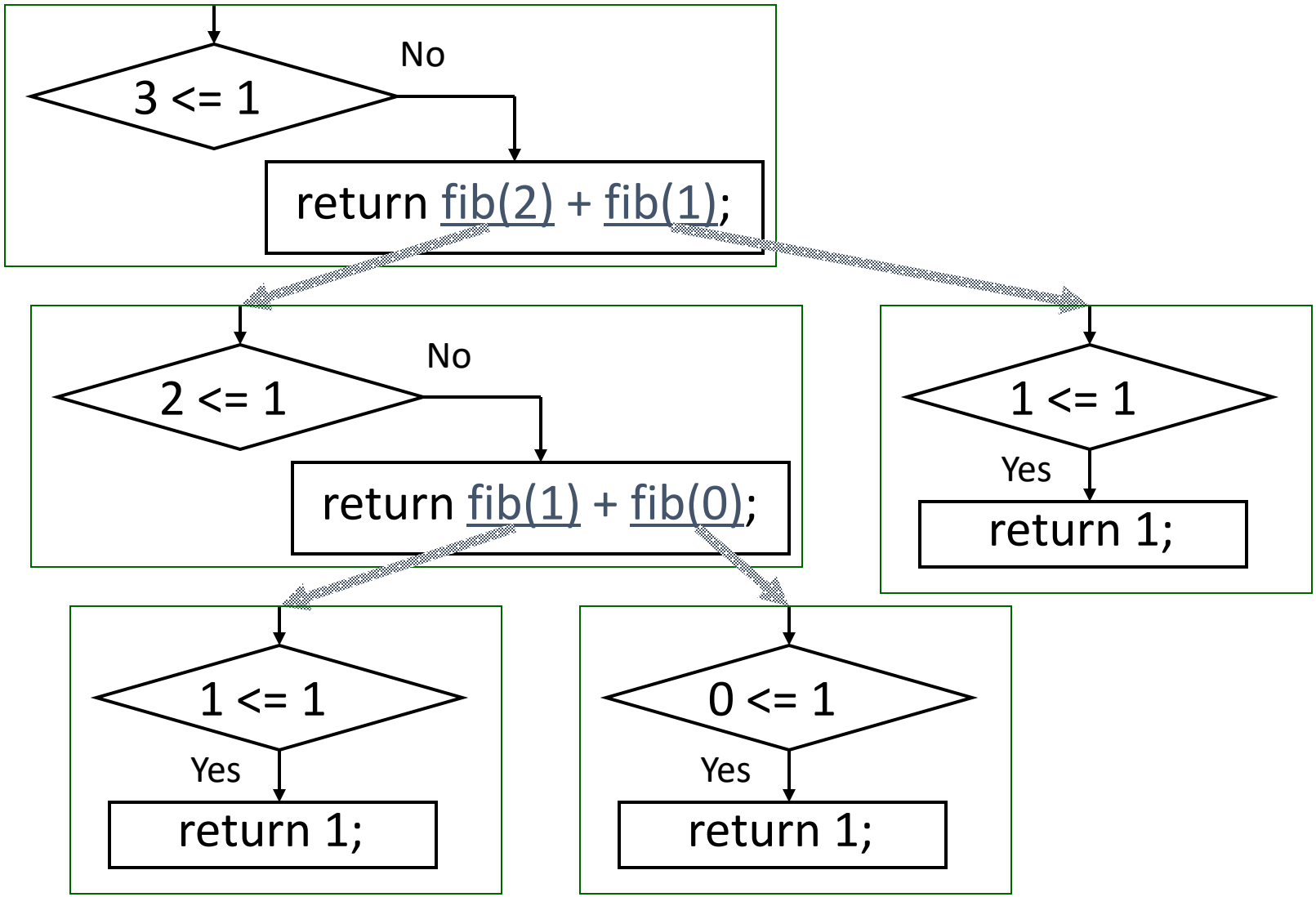
「再帰」によるフィボナッチ数列 ー n=2 のときの実行順 ー

n = 2 とすると



「再帰」によるフィボナッチ数列
ー n=3 のときの実行順 ー

n = 3 とすると



フィボナッチ数列の特性

- どれだけ大きな引数まで計算できるか調べてみよう.
- 引数を大きくするに従って，計算時間はどう変化するか調べてみよう.

例題２． McCarthyの91関数

- McCarthyの91関数

$$Mc(n)=\begin{cases} n-10 & \text{if } n>100, \\ Mc(Mc(n+11)) & \text{otherwise.} \end{cases}$$

の n 番めの数を計算するプログラムを，再帰関数を用いて作る.

```
#include <stdio.h>
#pragma warning(disable:4996)
int mc91(int n)
{
  if (n>100) {
    return n-10;
  }
  else {
    return mc91(mc91(n+11));
  }
}
int main()
{
  int n;
  int mc;
  printf("Enter a number: ");
  scanf("%d",&n);
  mc = mc91(n);
  printf("mc91(%d)=%d¥n",n ,mc );
  return 0;
}
```

McCarthyの91関数

実行結果の例

```
Enter a number: 20

mc91(20)=91
```

McCarthyの91関数の意義

- 本来のMcCarthyの91関数の定義は，再帰的な定義

- しかし，McCarthyの91関数の実際の値は単純

  ⇒　１００までの値を入れて，実際に計算させると，結果は「９１」

- McCarthyの91関数は，人工知能（プログラム理解，自動証明，記号処理など）の，良い例題とされてきた

課題１． McCarthyの91関数の特性

- いろいろな数に対して， **mc91** 関数の返す値を調べよ.
- 次の非再帰関数**m91**で，**mc91** 関数を置き換えて，同様に調べよ.

```
int m91(int n)
{
  if ( n>100 ) {
    return n-10;
  }
  else {
    return 91;
  }
}
```

例題３． Ackermann関数

- Ackermann関数

$$Ack(m,n)=\begin{cases} n+1 & if \quad m=0, \\ Ack(m-1,1) & if \quad n=0, \\ Ack(m-1,Ack(m,n-1)) & otherwise. \end{cases}$$

を計算するプログラムを，再帰関数を用いて書く．

```
#include <stdio.h>
#pragma warning(disable:4996)
int ack(int m, int n)
{
  if (m==0) {
    return n+1;
  }
  else if (n==0) {
    return ack(m-1,1);
  }
  else {
    return ack(m-1,ack(m,n-1));
  }
}
int main()
{
  int n;
  int m;
  int a;
  printf("m=");
  scanf("%d",&m);
  printf("n=");
  scanf("%d",&n);
  a = ack(m,n);
  printf("Ack(%d,%d)=%d¥n",m ,n ,a);
  return 0;
}
```

Ackermann関数

実行結果の例

m=2

n=4

Ack(2,4)=11

m=1 のときのAckermann関数

- アッカーマン関数の定義
  - Ack(0, n) = n +1
  - Ack(m, 0) = Ack(m-1, 1)
  - Ack(m, n) = Ack(m-1, Ack(m, n-1))
- m=1 とすると
  - Ack(1, 0) = Ack(0, 1) = 1 + 1 = 2
  - Ack(1, 1) = Ack(0, Ack(1, 0)) = Ack(0, 2) = 2 + 1 = 3
  - Ack(1, 2) = Ack(0, Ack(1, 1)) = Ack(0, 3) = 3 + 1 = 4

  数学的帰納法を使って，Ack(1, n) = n + 2 を証明することは簡単

Ackermann関数の再帰の回数

- m=0
  - Ack(0, 4) = 5　　再帰： 1回
  - Ack(0, 8) = 9　　再帰： 1回
  - Ack(0, 12) = 13　　再帰： 1回
- m=1
  - Ack(1, 4) = 6　　再帰： 10回
  - Ack(1, 8) = 10　　再帰： 18回
  - Ack(1, 12) = 14　　再帰： 26回
- m=2
  - Ack(2, 4) = 11　　再帰： 65回
  - Ack(2, 8) = 19　　再帰： 189回
  - Ack(2, 12) = 27　　再帰： 377回
- m=3
  - Ack(3, 4) = 125　　再帰： 10307回
  - Ack(3, 8) = 2045　　再帰： 2785999回

（関数 ack を呼び出した回数）

Ackermann関数の意義

- Ackermann関数のプログラムは，「関数の再帰呼び出し」を使って，書くことができた
- しかし，mが少し大きくなると，Ackermann関数の値が爆発（つまり再帰の回数も爆発）して，事実上，計算不可能
- しかも，Ackermann関数は，「普通の繰り返し文では書けない」ことが証明されている（このことを，「原始帰納的でない」という）

課題２．Ackermann関数の特性

- どれだけ大きな引数まで計算できるか調べよ.
- 引数を大きくするに従って，計算時間はどう変化するか調べよ.

課題３

- フィボナッチ数列，McCarthyの91関数，Ackermann関数で，return文の直前で戻り値を表示させ，計算の様子を調べよ.

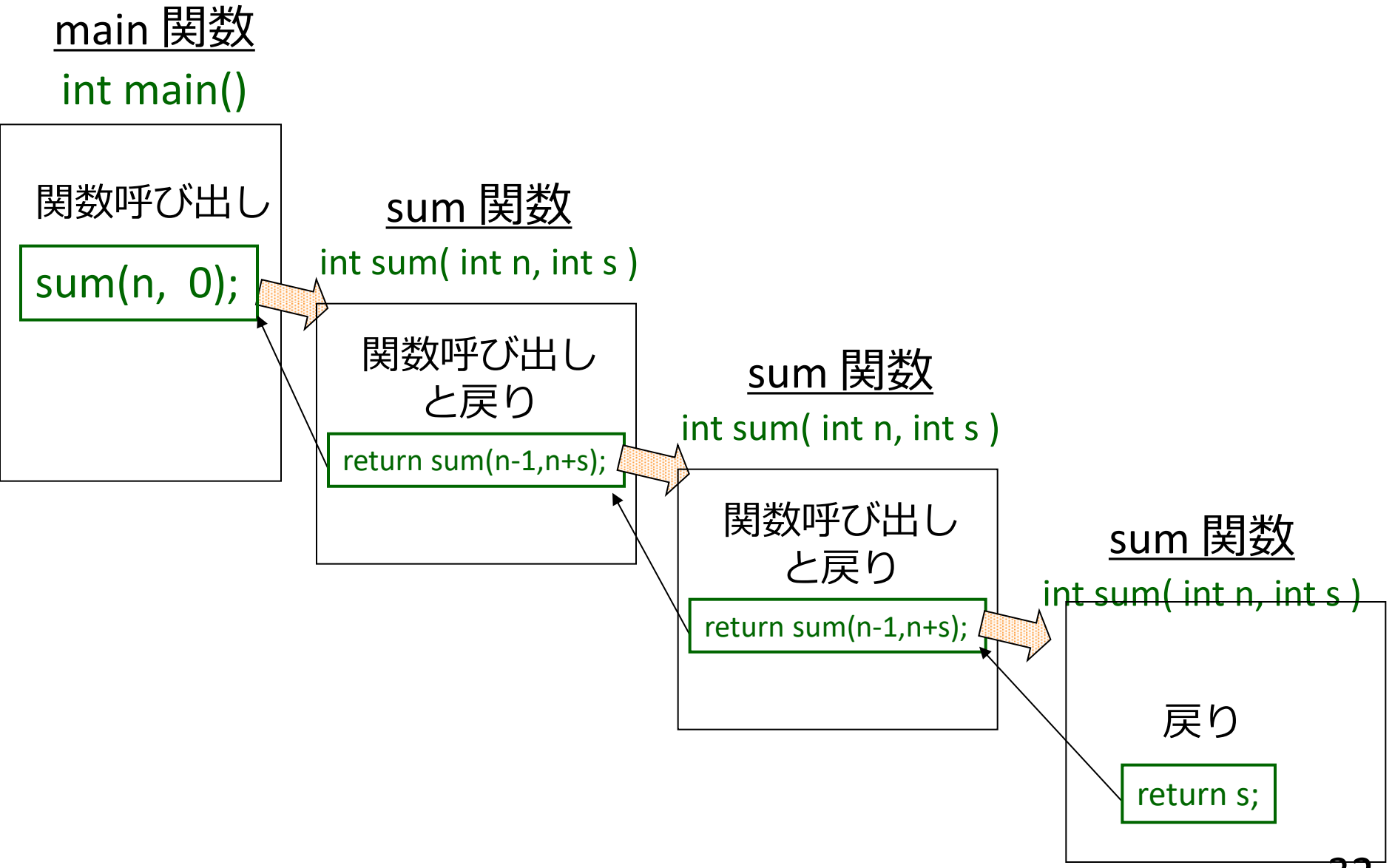
例題４．総和を求める末尾再帰関数

- 総和を求める関数を，末尾再帰関数の形で書く
  - 末尾再帰関数については，次ページ以降で説明

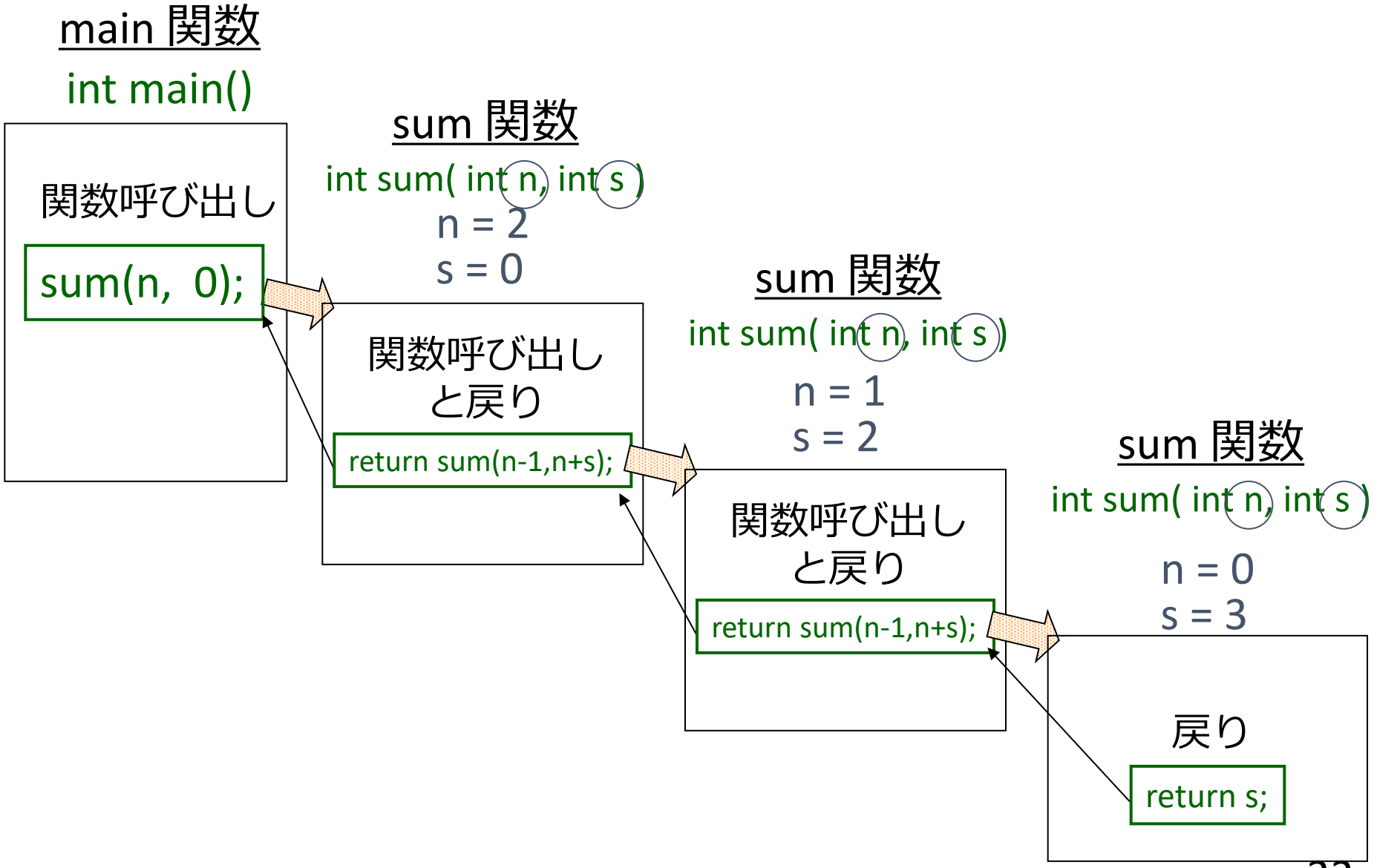
```
#include <stdio.h>
#pragma warning(disable:4996)
int sum(int n, int s)
{
  if (n==0) {
    return s;
  }
  else {
    return sum(n-1,n+s);
  }
}
int main()
{
  int n;
  int souwa;
  printf("Enter a number: ");
  scanf("%d",&n);
  souwa = sum(n, 0);
  printf("sum(%d)=%d¥n",n, souwa);
  return 0;
}
```

return 文の中でのみ再帰呼び出し

関数呼び出しの流れ
（main 関数で n = 2 のとき）



n と s の値の変化
（main 関数で n = 2 のとき）



末尾再帰関数とは

- 再帰関数の中で，一番最後（つまり return 文など）で，ただ１度だけ，その本体の再帰呼び出しを行うもの
  - 再帰呼び出しを行ったら，その関数がすぐに終わる
  - 再帰呼び出しの結果（戻り値）を得たら，その関数自体の戻り値として，直接戻す

- 「末尾再帰関数」と，「繰り返し文を使ったプログラム（再帰関数の無いもの）」とは，互いに，簡単に書き直せる

再帰関数と末尾再帰関数

- 再帰関数で，末尾再帰関数に書き直すことができるもの
  - 整数の総和，階乗，フィボナッチ数列など
- 再帰関数で，末尾再帰関数に書き直すことができないもの
  - Ackermann関数など

末尾再帰の計算の方法

- 途中結果を引数に与えて，関数の再帰呼び出しを行う.
- 最後の再帰呼び出しでは，与えられた途中結果を最終結果として返す

例）

2+0

→ s(2,0)　↓　1+2

→ s(1,2)　↓

→ s(0,3)

→ 3

課題４

次の階乗関数を，末尾再帰の形に書き直せ．また，書き直した階乗関数を使う main 関数を書き，正しく動作することを確認すること．

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

課題４について

- 末尾再帰で無い例：

  return n * factorial(n-1);

  return factorial(n-1) * n;

  いずれも，factorial(n-1) の返り値を使って，n との掛け算を行っている
- 末尾再帰にするには：

  途中結果を引数に含めること

  例）return factorial(n-1, n*s);

課題 5

- フィボナッチ数列を末尾再帰の形で書け.

ヒント： 一歩前の途中結果 f_{k-1} と，二歩前の途中結果 f_{k-2} の二つを引数に加えて，再帰呼び出しする.