第6回C言語勉強会

～関数の復習と再帰関数～

復習

関数

プロトタイプ宣言

```
int max(int a, int b);

int main(void)
{
        printf("%d¥n", max(10,20 ));
        return 0;
}

int max(int a, int b){
        if (a>b) return a;
        else return b;
}
```

関数呼び出し

関数の中身

関数の定義

関数名

引数の型

```
int max(int a, int b){
        if (a>b) return a;
        else return b;
}
```

戻り値の型

戻り値

この関数は大きい方の引数を戻り値で返す
つまり実行すると……

関数の定義

関数名

引数の型

```
int max(int a, int b){
        if (a>b) return a;
        else return b;
}
```

戻り値の型

戻り値

この　返す

```
20
続行するには何かキーを押
```

問題

- 現在消費税が8%になり、様々なものの値段が上がった。中には、増税分以上の値上げをしているものもある(便乗値上げ)
- そこで、消費税5%と8%での商品の値段(税込み)を入力すると、いくら便乗値上げをしているかを明らかにするプログラムを作りたい
- 入力を5%と8%での金額
- 出力を便乗値上げした金額　とする
- 自分で知っている金額を色々と試してみよう

入力例

- Dトール　200円から２２０円に値上げ
- Tリーズ　300円を310円に

- Y野家 280円を300円に

- 自動販売機　120円から130円に値上げ

値段はあくまで参考であり、特定の企業等を誹謗中傷するものではありません

解答例

```
#include <stdio.h>
double neage(double pre, double post);
int main(void)
{
        printf("便乗値上げ分は%.0f円です¥n", neage(280,
300));
        return 0;
}
double neage(double pre, double post){
        printf("値上げ前の税抜き価格:%.0f¥n", pre / 1.05);
        printf("値上げ後の税抜き価格:%.0f¥n", post / 1.08);
        double ret;
        ret = post / 1.08 - pre / 1.05;
        return ret;
}
```

再帰関数

関数内で自分自身の関数を呼び出すことを
“再帰呼出し”という

```
void hello(int i){
        if (i==0) return;
        printf("Hello!");
        hello(--i);
}
```

どのような仕組みになっているか

再帰関数

関数内で自分自身の関数を呼び出すことを
“再帰呼出し”という

C:¥WINDOWS¥system32¥cmd.exe

```
Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!
続行するには何かキーを押してください . . .
```

どのような仕組みになっているか

再帰関数の仕組み

```
void hello(int i){

        if (i==0) return;

        printf("Hello!");

        hello(--i);
}
```

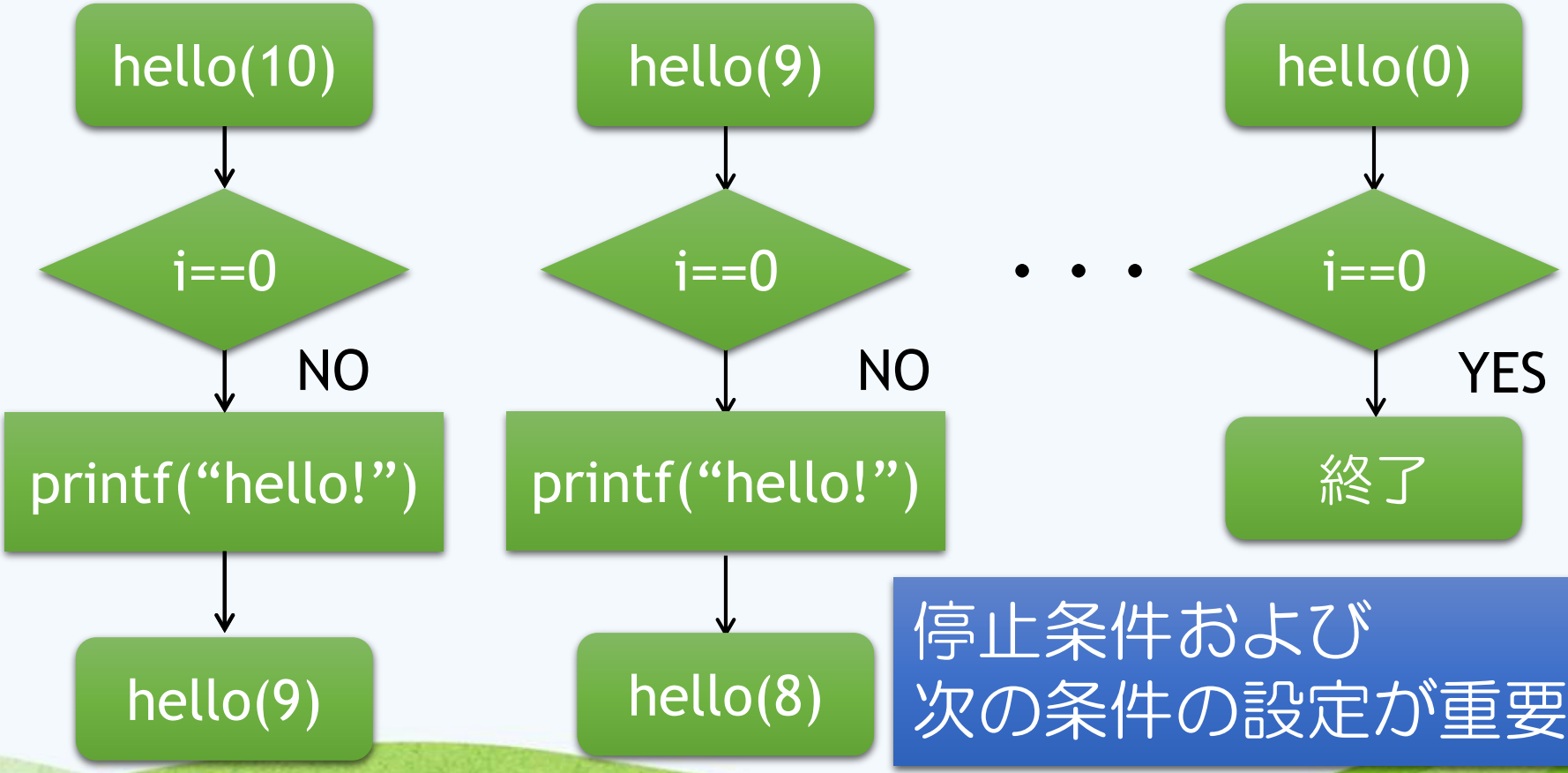
停止条件

処理の内容

次に与える条件

再帰関数の仕組み

- main関数で呼び出す



停止条件および
次の条件の設定が重要

再帰関数の仕組み

- 数字を順にカウントしていくプログラムを考える

下の2つのプログラムをそれぞれ動かしてみて、
動き方の違いを確かめてみる

```
void count(int cnt){
        if (cnt == 0) return;
        printf("%d¥n", cnt);
        count(--cnt);
}
```

```
void count(int cnt){
        if (cnt == 0) return;
        count(--cnt);
        printf("%d¥n", cnt);
        }
```

再帰関数の仕組み

- 数字を順にカウントしていくプログラムを考える

下の2つのプログラムをそれぞれ動かしてみて、
動き方の違いを確かめてみる

```
void c
                    return;
                    n", cnt);
                    ;
}
```

```
0
1
2
3
4
5
6
7
8
9
```

```
void cou
                    return;
                    , cnt);
```

```
10
9
8
7
6
5
4
3
2
1
```

練習問題

- 再帰関数を使って階乗の計算をしてみる
- 入力としてint型の数字を得る。
- 出力としてその階乗を出す。
- ヒント：停止条件をどうするか？
  戻り値に再帰呼出しを行えば良さそう

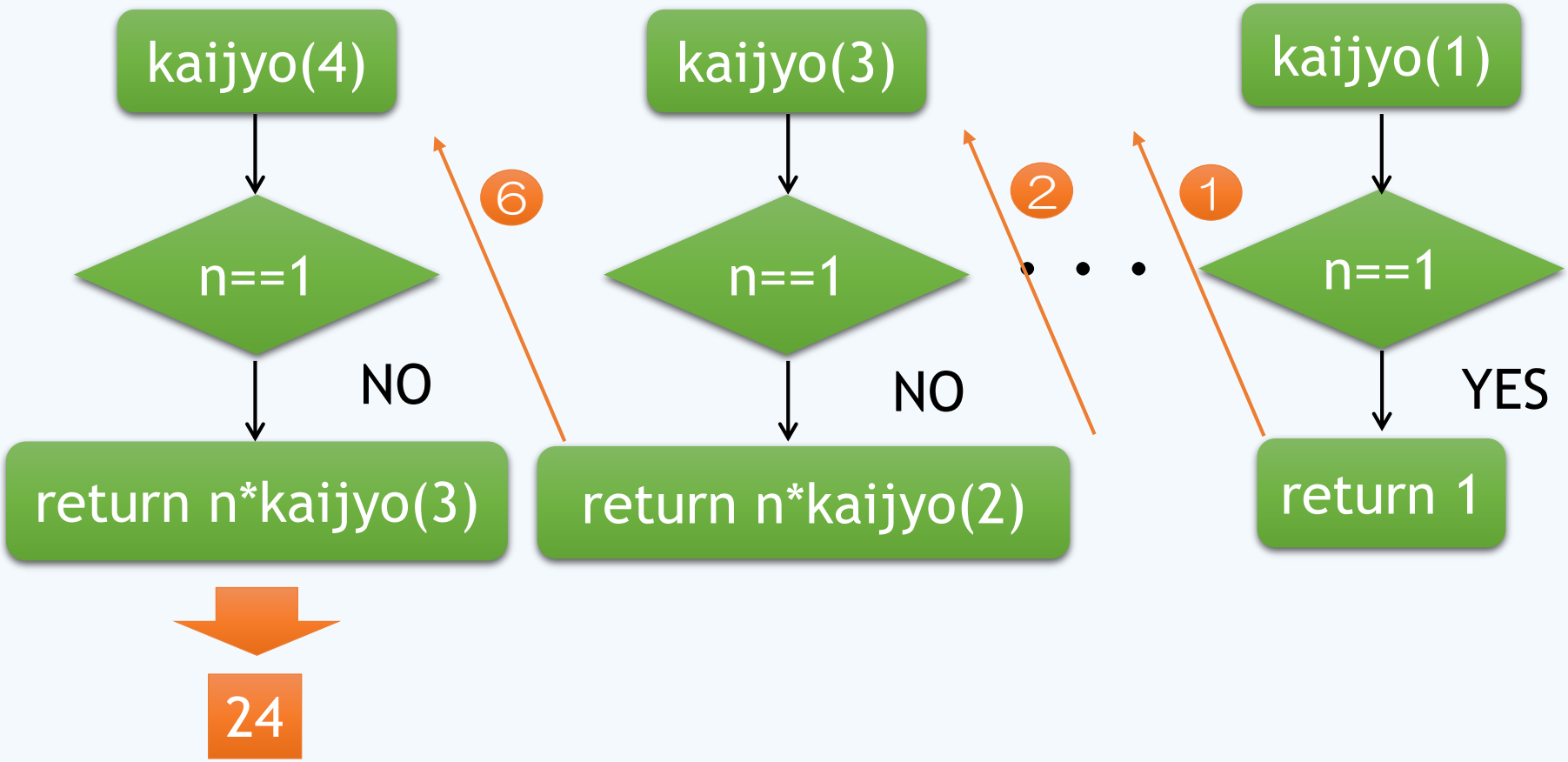
```
int kaijo(int n){
        if (n == 1) return 1;
        return n*kaijo(n - 1);
}
```

練習問題

- 再帰関数を使って階乗の計算をしてみる
- 入力としてint型の数字を得る。
- 出力としてその階乗を出す。
- ヒント：停止条件をどうするか？
  戻り値に再帰呼出しを行えば良さそう

```
int kaijyo(int n){
        if (n == 1) return 1;
        return n*kaijyo(n - 1);
}
```

再帰の動き



再帰関数は塗りつぶし処理に使われていたりする

問題：再帰関数を使って
フィボナッチ数列を書いてみよう

フィボナッチ数？

フィボナッチ数とは、

i = １ の時 Fibo(1) = 1
i = ２ の時 Fibo(2) = 1
i = ３ の時 Fibo(3) = 2
i = ４ の時 Fibo(4) = 3
i = ５ の時 Fibo(5) = 5
－－－－－－
i = n の時 Fibo(n) = Fibo(n－1) + Fibo(n－2)
－－－－－－
Fibo(1)、Fibo(2)を例外として、任意の項は
その前の２つの項の合計になる。

兎のつがいの問題

- 1つがいの兎は、産まれて2か月後から毎月1つがいずつの兎を産む。
- 兎が死ぬことはない。
- この条件のもとで、産まれたばかりの1つがいの兎は1年の間に何つがいの兎になるか？

	産まれたばかりのつがい	生後1か月のつがい	生後2か月以降のつがい	つがいの数（合計）
0か月後	1	0	0	1
1か月後	0	1	0	1
2か月後	1	0	1	2
3か月後	1	1	1	3
4か月後	2	1	2	5
5か月後	3	2	3	8
6か月後	5	3	5	13
7か月後	8	5	8	21
8か月後	13	8	13	34
9か月後	21	13	21	55
10か月後	34	21	34	89
11か月後	55	34	55	144
12か月後	89	55	89	233

つがいの合計は1、1、2、3、5、8、13、21、34……
フィボナッチ数列となっている

問題:再帰関数を使って フィボナッチ数列を表示させよう

- フィボナッチ数列の定義は
任意の項の値は、その2つ前の項の合計となる
Fibo(n) = Fibo(n - 1) + Fibo(n - 2)

- プログラム上ではどのように書けばよいか？
……関数fibを定義し、再帰呼出しをすればよい

```
int fib(int n)
{
        if (n == 0 || n == 1) return 1;
        return fib(n - 1) + fib(n - 2);
}
```

問題:再帰関数を使って フィボナッチ数列を表示させよう

- フィボナッチ数列の定義は

任意の項の値は、その2つ前の項の合計となる

Fibo(n) = Fibo(n - 1) + Fibo(n - 2)

- プログラム上ではどのように書けばよいか？

……関数fibを定義し、再帰呼出しをすればよい

```
int fib(int n)
{
        if (a == 0 || a == 1) return 1;
        return fib(n - 1) + fib(n - 2);
}
```

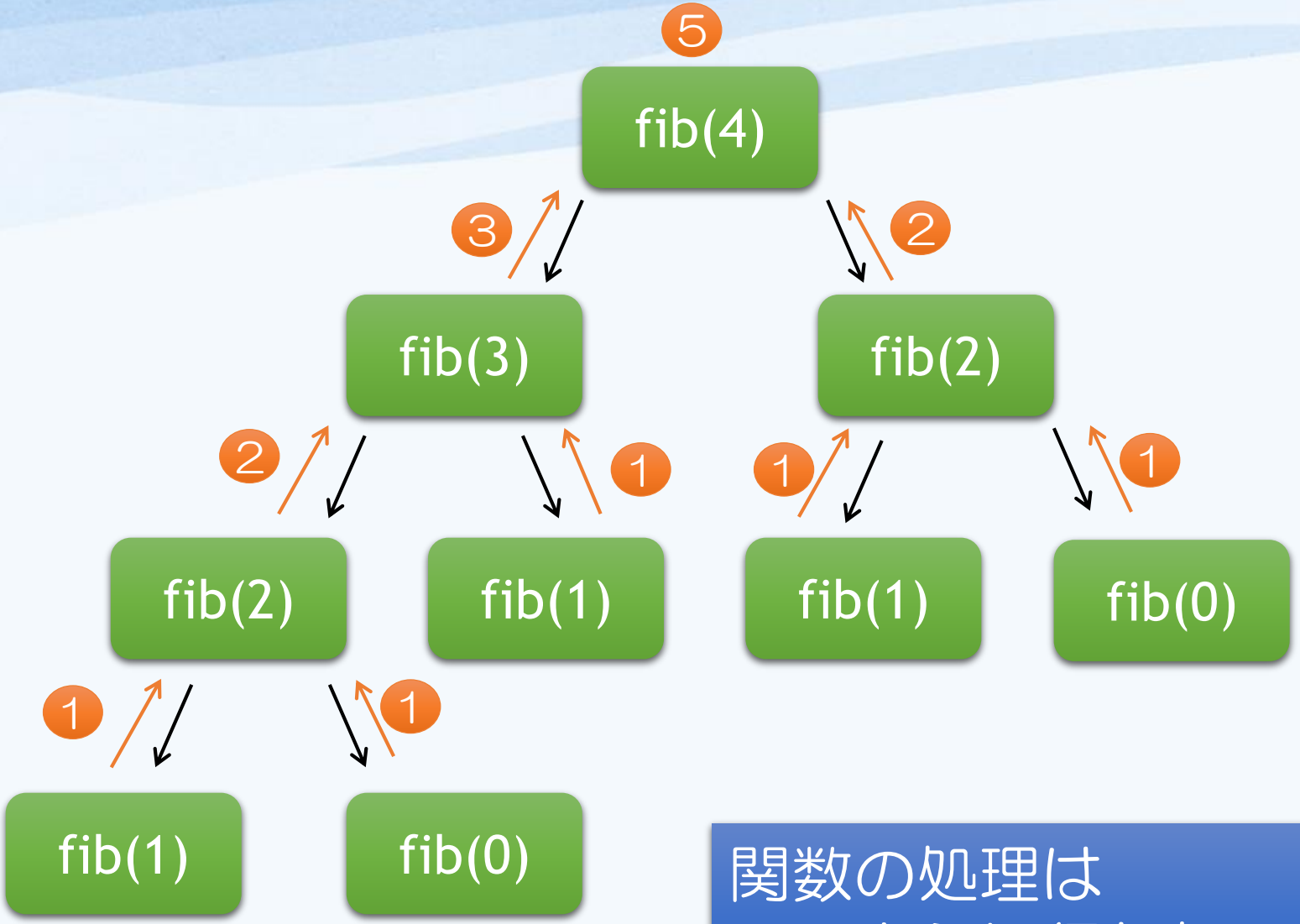
fib関数

```
int fib(int n)
{
    if (a == 0 || a == 1) return 1;

    return fib(n - 1) + fib(n - 2);
}
```

引数の値が0か1なら終了
戻り値として1を返す

戻り値として
前2つの項の和を返す



関数の処理は
このように行われている

発展問題

- もっと効率よくフィボナッチ数列を求める為に

1. メモ化再帰

2. 動的計画法

という方法があるので説明を載せる

メモ化再帰

- 1度計算をしたfib関数の値を保持することで、繰り返し計算を行わずに済む。

fib(4)=fib(3)+fib(2)

=(fib(2)+fib(1))+(fib(1)+fib(0))

=((fib(1)+fib(0))+fib(1))+(fib(1)+(fib(0))

この場合fib(2)が2回計算していて無駄になる

そこでfib関数の結果を配列に保存し、
一度計算したものを記憶させればよい

メモ化再帰

- 実装例

```
int memo[50] = { 1, 1 };

int fib2(int n)
{
        if (memo[n] !=0) return memo[n];
        else{
                return memo[n] = fib2(n - 1) + fib2(n - 2);
        }
```

メモ化再帰

- どのように動くかというと……

1. 計算済みであれば

memo[n]の値が0以外が入っているはずなので
その値を戻り値として返す

2. 計算がまだならば

memo[n]は初期値0のままなので戻り値として
fib(n-1)+fib(n-2)を返し、さらにmemo[n]に
その値を格納する

動的計画法

- この方法では再帰は使わないが、
高速なプログラムを実装できる

- この方法では、

1. まずフィボナッチ数列の値を格納する配列を作り、その値を入れていく。
2. 次の項を求めるときにはその配列に格納されている値を使う。

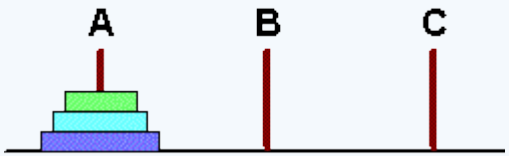
動的計画法

- 実装例

```
int result[50];
int fib3(int a){
        int i;
        result[0] = 1;
        result[1] = 1;
        for (i = 2; i <= a; i++){
                result[i] = result[i - 1] + result[i - 2];
        }
        return result[a];
}
```

暇な人は

- 再帰を使うハノイの塔という問題がある



1. 一回に一枚の円盤しか動かしてはいけません。
2. 移動の途中で円盤の大小を逆に積んではいけません。常に大きい方の円盤が下になるようにして下さい。
3. 棒A，B，C以外のところに円盤を置いてはいけません。

引用元:http://ews2.cc.niigata-u.ac.jp/~takeuchi/tbasic/BackGround/Hanoi.html