# プログラミング応用
# http://bit.ly/kosen02

Week10
再帰アルゴリズム第2回

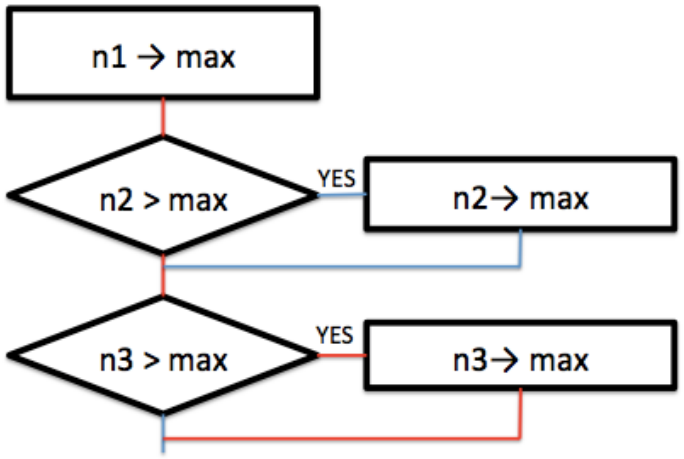
# 本日の内容

- 再帰アルゴリズムの動作を解析
  - トップダウン解析
  - ボトムアップ解析
- 代表的な再帰アルゴリズム
  - ユークリッドの互除法(最大公約数)
- 演習
  - 再帰アルゴリズムの解析

# アルゴリズムの定義(復習)

問題を解くためのものであって、明確に定義され、
順序付けられた有限個の規則からなる集合

例）三値の最大値を
求めるという問題



# 再帰アルゴリズム(概要)

- 再帰
  - 関数の中で自分自身を呼び出す
  - 記述が簡潔になるが動作を理解するのは難しい
- 再帰アルゴリズム
  - 再帰を用いてより簡潔に
    (しばしば効率的に)記述されたアルゴリズム
  - 関数の中で自身と同じ関数を2回以上呼び出す
    再帰アルゴリズムを真に再帰的であるという
  - 特に「真に再帰的」の場合アルゴリズムの
    挙動を負うのは大変

# 真に再帰的なアルゴリズム（例）

- フィボナッチ数を求めるアルゴリズム (week9/fibo_recursive.c)

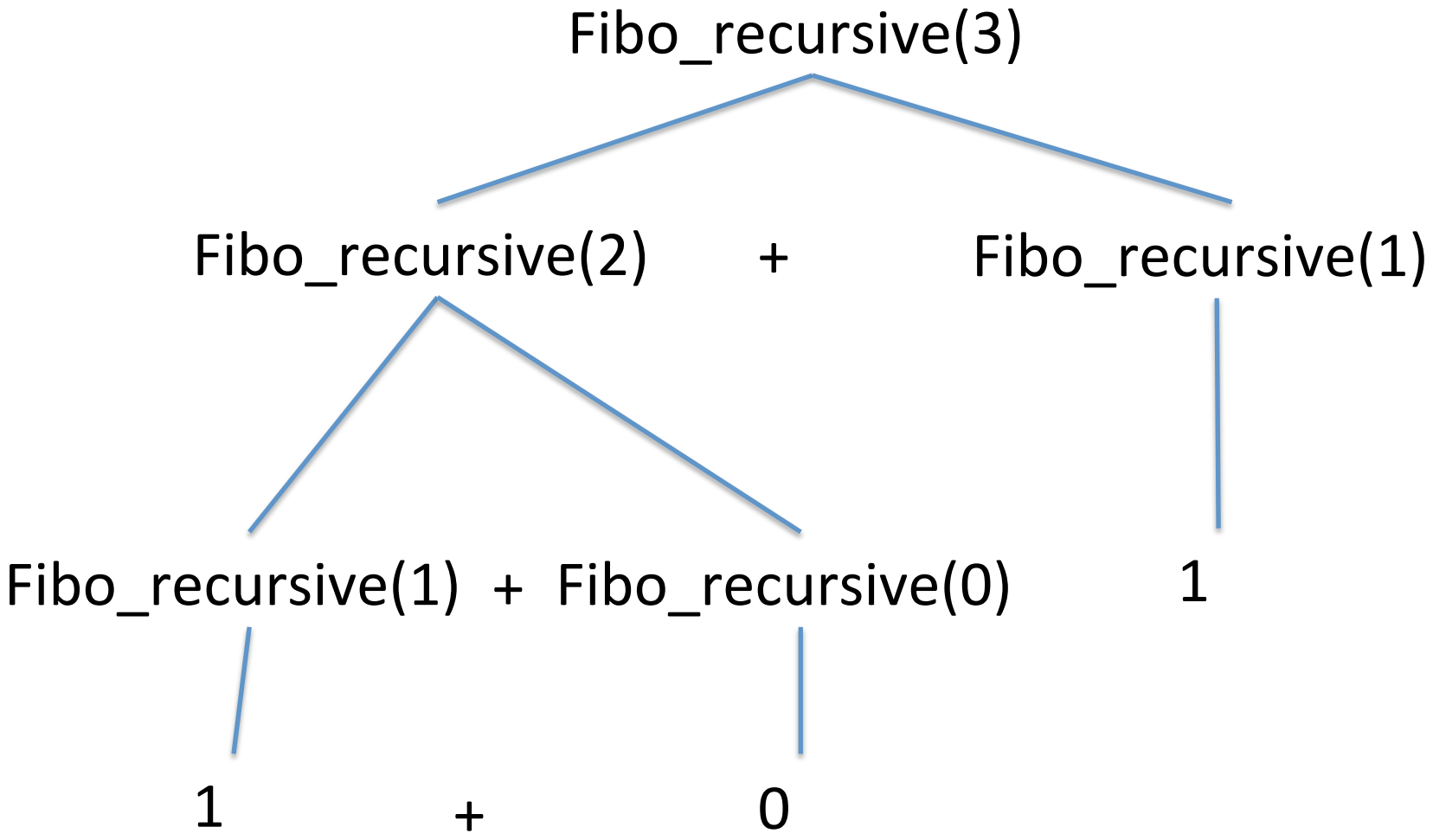
```
 3 int fibo_recursive(int x) {
 4   int f = 0;
 5   switch (x) {
 6     case 0:
 7       f = 0; break;
 8     case 1:
 9       f = 1; break;
10     default:
11       f = fibo_recursive(x - 1) + fibo_recursive(x - 2);
12       break;
13   }
14     return (f);
15 }
```

# 2つの解析方法

- トップダウン解析
  - fibo_recursive(3)の挙動を調べ、fibo_recursive(2)、fibo_recursive(1)と下がっていく解析方法
  - 図を書いて可視化する
- ボトムアップ解析
  - fibo_recursive(0)の挙動を調べ、fibo_recursive(1)、fobo_recursive(2)と上がっていく解析方法
  - 図は書かなくてもよい

※どちらの解析も紙とペンを使って行うと良い

# 例）トップダウン解析


Fibo_recursive(3)
Fibo_recursive(2) + Fibo_recursive(1)
Fibo_recursive(1) + Fibo_recursive(0) 1
1 + 0

# 例）ボトムアップ解析

- Fibo_recursive(3)を求める場合
  1. Fibo_recursive(0)を求める
     → 0
  2. Fibo_recursive(1)を求める
     → 1
  3. Fibo-recursive(2)を求める
     → Fibo_recursive(0) + Fibo_recursive(1)
     = 0 + 1 = 1
  4. Fibo_recursive(3)を求める
     → Fibo_recursive(2) + Fibo_recursive(1)
     = 1 + 1 = 2

# 本日の内容

- 再帰アルゴリズムの動作を解析
  - トップダウン解析
  - ボトムアップ解析
- 代表的な再帰アルゴリズム
  - ユークリッドの互除法(最大公約数)
- 演習
  - 再帰アルゴリズムの解析

# 最大公約数

- 最大公約数(Greatest common divisor)
  - 例) 8と22の最大公約数は2
    （8も22も2で割り切ることが出来る）
  - 皆さんだったら
    どのように最大公約数を求めますか？

# 最大公約数を求めるアルゴリズム

1. しらみつぶしに正解を探す
2. 因数分解
3. ユークリッドの互除法

# しらみつぶしに探すプログラム

```c
/* しらみつぶしに最大公約数を探すプログラム */
#include <stdio.h>

int main(void) {
  // 最大公約数を求める2つの数を指定
  int x=8, y=22;
  // xの方が小さな数になるようにする
  if( x > y ) { tmp = x; x = y; y = tmp; }
  // 最大公約数を探す
  for(int i=x; i>0; --i) {
    if((x % i)==0 && (y % i) == 0) {
      printf("最大公約数: %d\n", i);
      break;
    }
  }
}
```

# しらみつぶしに探すプログラム

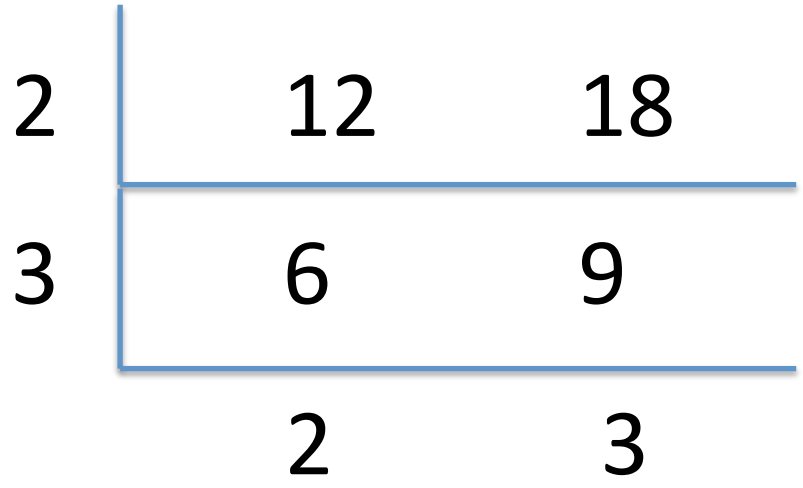
```
 1 /* しらみつぶしに最大公約数を探すプログラム */
 2 #include <stdio.h>
 3
 4 int main(void) {
 5   // 最大公約数を求める2つの数を指定
 6   int x=8, y=22;
 7   // xの方が小さな数になるようにする
 8   if( x > y ) { tmp = x; x = y; y = tmp; }
 9   // 最大公約数を探す
10   for(int i=x; i>0; --i) {
11     if((x % i)==0 && (y % i) == 0) {
12       printf("最大公約数: %d\n", i);
13       break;
14     }
15   }
16 }
```

大きな数だと
比較回数が増大

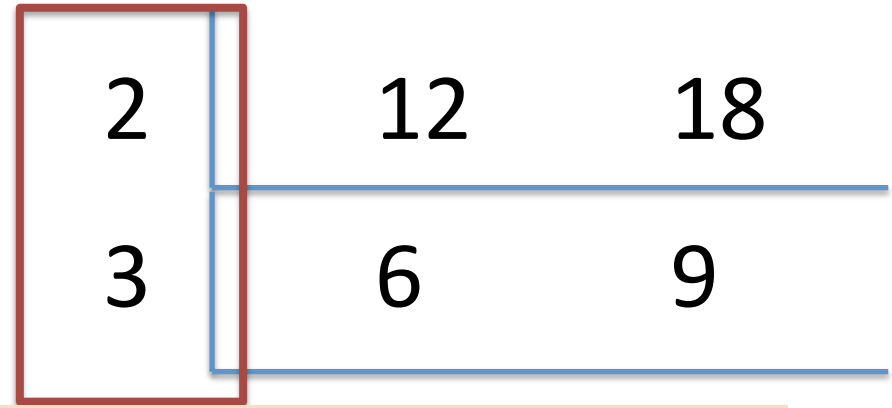
# 因数分解

# 因数分解

# 因数分解

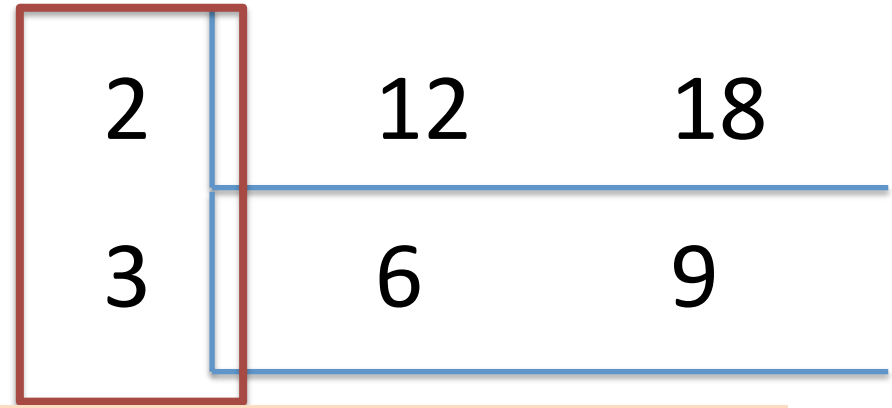


# 因数分解




最大公約数 = 2 × 3 = 6

# 因数分解




最大公約数 = 2 × 3 = 6

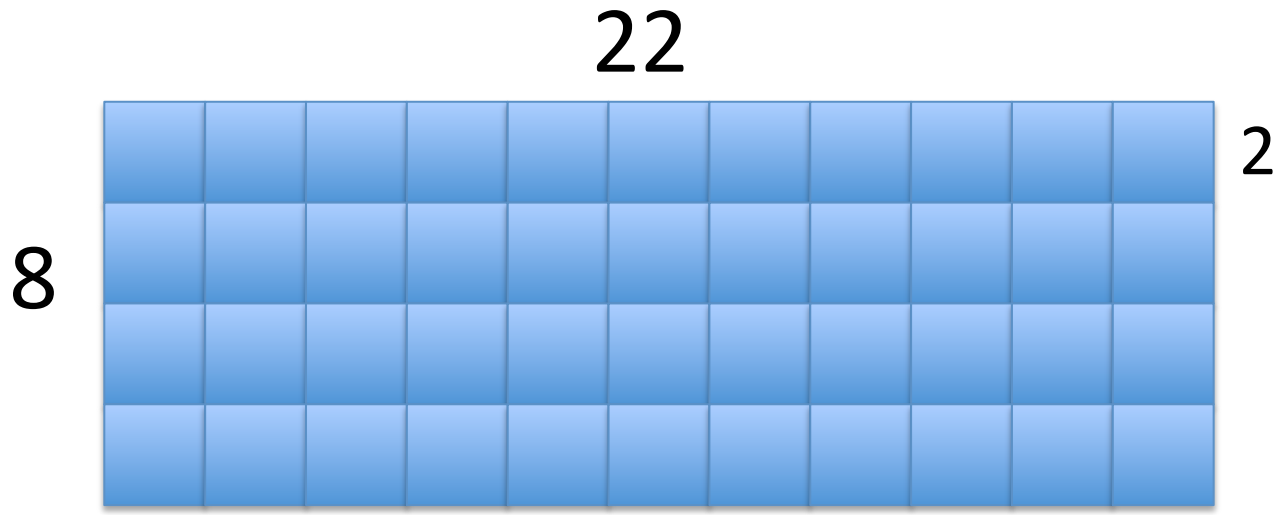
素因数分解できれば最大公約数が求まるが、
桁数の多い素数を素因数分解する
効率的な方法は見つかっていない

# ユークリッドの互除法

- ユークリッドの互除法
  - 最大公約数を求めるための再帰アルゴリズム
  - 簡潔で速いアルゴリズム
    （先週の良いアルゴリズムの要件を参照）

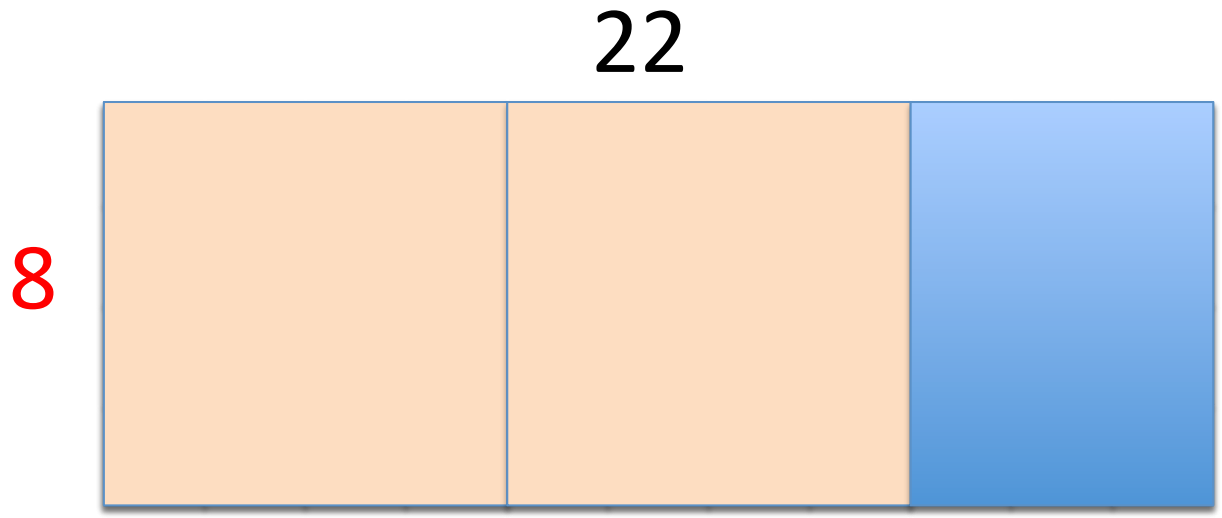
# ユークリッドの互除法(アイデア)

- 最大公約数とはなにか？
  - ユークリッドの互除法は最大公約数を求める問題を図形で考える
  - 22×8の長方形を正方形で埋め尽くすことを考える
  - 長方形を埋め尽くすことのできる正方形の最大の辺の長さが最大公約数



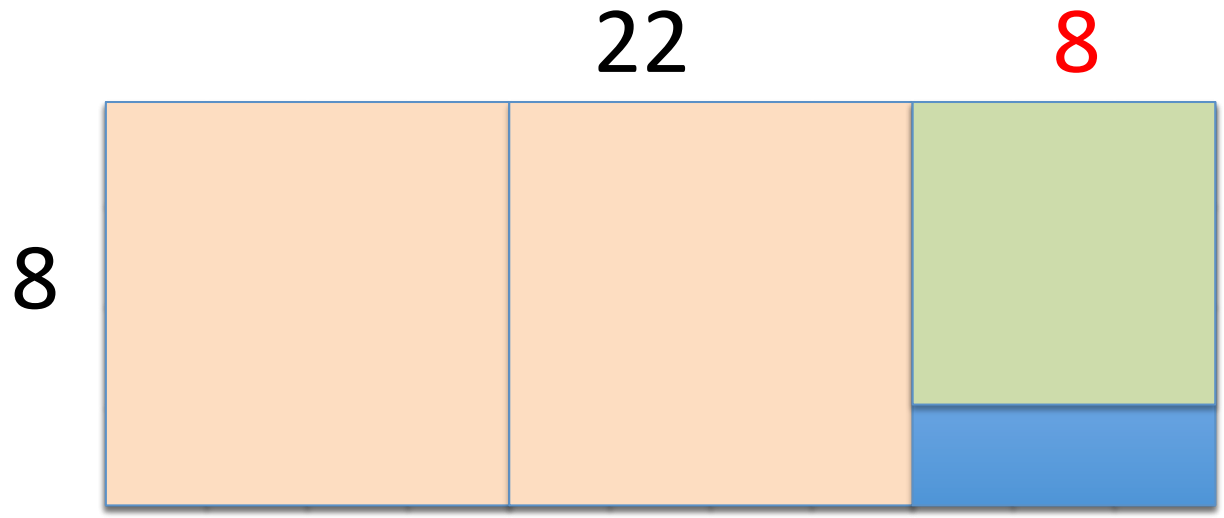
# ユークリッドの互除法(アルゴリズム)

- ステップ1.
  短い方の辺の長さを一辺とする正方形で
  長方形を埋める



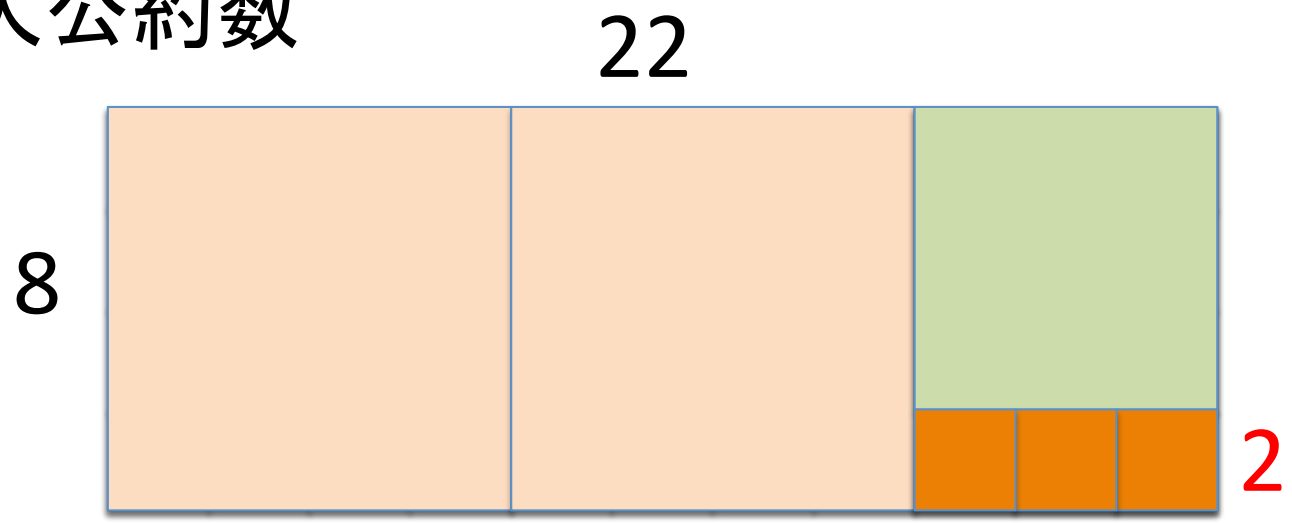
# ユークリッドの互除法(アルゴリズム)

- ステップ2
  埋められなかった長方形の
  短い方の辺の長さを一辺とする正方形で
  長方形を埋める



# ユークリッドの互除法(アルゴリズム)

- ステップ3
  さらに埋められなかった長方形の
  短い方の辺の長さを一辺とする正方形で
  長方形を埋める
  → 全て埋まったときの一辺の長さが
  最大公約数



# C言語での表現(1/2)

- ユークリッドの互除法がやっていること
  1. 2つの整数が与えられたとき大きい方の数を小さい方の数で割ってみて、割り切れるならば小さい方の数が最大公約数
  2. 割り切れない場合は小さい方の数と得られた余りに対して同じ手続きを繰り返す

# C言語での表現(2/2)

```c
#include <stdio.h>

int gcd(int x, int y) {
  if (y == 0) {
    return x;
  } else {
    return gcd(y, x % y);
  }
}

int main(void) {
  printf("8と22の最大公約数：%d\n", gcd(8, 22));
}
```

# C言語での表現(2/2)

```c
#include <stdio.h>

int gcd(int x, int y) {
  if (y == 0) {
    return x;
  } else {
    return gcd(y, x % y);
  }
}

int main(void) {
  printf("22と8の最大公約数：%d\n", gcd(22, 8));
}
```

# トップダウン解析(ユークリッドの互除法)

- gcd(22, 8)
  - y != 0なのでgcd(8, 22 % 8) → gcd(8, 6)
- gcd(8, 6)
  - y != 0なのでgcd(6, 8 % 6) → gcd(6, 2)
- gcd(6, 2)
  - y != 0なのでgcd(2, 6 % 2) → gcd(2, 0)
- gcd(2, 0)
  - y == 0なので2を最大公約数として出力

# 余談：ユークリッド

- BC330-275
- 幾何学について著した『原論』が有名
- 素数x素数=素数になることを証明
- 巨大な数の素因数分解が困難なことを認識
  → 互除法の発明
- 互除法も原論に記されている
  → 世界最古のアルゴリズム

# 演習1(再帰アルゴリズムの解析)

1. 紙とペンを用意
2. 次項のプログラム(recursive.c)においてmain関数からrecursive(4)を呼び出した際の動作について、図を書きながらトップダウン解析しなさい。
3. 図は書かずにボトムアップ解析し、buttomup_analysis.txtというファイル名で解析結果を記述し提出しなさい

```c
#include <stdio.h>

void recursive(int n) {
  if (n > 0) {
    recursive(n-1);
    printf("%d\n", n);
    recursive(n-2);
  }
}

int main(void) {
  recursive(4);
}
```

# 演習2(ユークリッドの互除法)

- ユークリッドの互除法で6と22の最大公約数を求める場合のアルゴリズムの動作を図形で表現しなさい
（提出不要です）

# 演習3-0(しらみつぶし)

- しらみつぶしに最大公約数を求めるプログラム(gcd1.c)を記述しなさい

# 演習3-1(ユークリッドの互除法)

- ユークリッドの互除法をC言語のプログラム(gcd2.c)を記述し動作を理解しなさい
  (講義資料のプログラムを記述すれば良い)

# 演習4(ユークリッドの互除法)

- ユークリッドの互除法を再帰を用いずにC言語で記述しなさい。
  - ファイル名はgcd3.c
  - 講義資料の最初の方に示しているのは「すべての数についてしらみつぶしに最大公約数を探す方法」。この演習ではユークリッドの互除法の動作をwhile文等を用いて記述してみてください。

# 演習Extra(早く終わった人向け)

- 「ハノイの塔」について調べ、どのような問題かまとめなさい
- ハノイの塔のアルゴリズムを調べ動作を確認しなさい

# 次回

- 「探索アルゴリズム」について学習します