基本的なアルゴリズムと，再帰

基本的なアルゴリズム

　　階乗，　順列・組み合わせ，

　　最大公約数，　素数

再帰関数（recursive function）

　　自分自身を呼び出す関数

再帰プログラミング（recursive programming）

　　関数が自分自身を呼び出すプログラミングスタイル

再帰呼び出し（recursive call）

　　再帰関数の呼び出し

階乗（factorial）　ｎ！

求め方（1）

```
fact = 1
fact = n * fact                 fact = (n - 1) * fact
fact = (n - 2) * fact           ・・・
fact = 2 * fact                 fact = 1 * fact
```

求め方（2）

０！　＝　１

ｎ！　＝　ｎ　×　（ｎ－１）！　　漸化式

ｎの階乗を返す関数 fact_r (n) に対し，

```
fact_r (0)  ⇒  1
fact_r (n)  ⇒  n * fact_r (n - 1)
```

```c
#include <stdio.h>

#define    N_MAX    13

/* 繰り返しによる定義 */

int fact_I (int n)
{
  int fact = 1;

  while (0 < n) {
    fact *= n;
    n --;
  }
  return fact;
}
```

```c
/* 再帰による定義 */

int fact_r (int n)
{
  if (n == 0)    /* 再帰の終了 */
    return 1;    /* 条件 */
  else
    return n * fact_r (n - 1);
}
```

```
int main(void)
{
  int i;

  for (i = 0; i <= N_MAX; i++) {
    printf ("%d, %d, %d ¥n",
          i, fact_l(i), fact_r(i));
  }
}
```

```
0, 1, 1
1, 1, 1
2, 2, 2
3, 6, 6
4, 24, 24
5, 120, 120
6, 720, 720
7, 5040, 5040
8, 40320, 40320
9, 362880, 362880
10, 3628800, 3628800
11, 39916800, 39916800
12, 479001600, 479001600
13, 1932053504, 1932053504
```

順列（permutation）

nPr ＝ ｎ！／（ｎ－ｒ）！

組み合わせ（combination）

nCr ＝ ｎ！／（ｒ！ （ｎ－ｒ）！）

求め方（1）

ｎ！ と ｒ！ と（ｎ－ｒ）！ を求めて，乗除．

求め方（2）

nC0 ＝ 1

nCr ＝ （ｎ－ｒ＋１）nCr-1 ／ｒ

/* 整数演算．割り算は最後 */

```c
#include <stdio.h>

# define N 12
# define R  4

int comb_r (int n, int r)
{
  if (r == 0)
    return 1;
  else
    return (n - r + 1)
    * comb_r(n, r-1) / r;
}

int fact_r (int n)
{
  if (n  == 0)
    return 1;
  else
    return n * fact_r (n - 1);
}
```

```c
int main(void)
{
  int n, r;

  n = N;
  r = R;

  printf ("%dC%d: %d %d ¥n",
    n, r,
    fact_r(n) / (fact_r(r)
    * fact_r(n - r)),
    comb_r(n, r));
}
```

```
12C4: 495 495
```

最大公約数（GCD：　greatest common divisor (measure)）

ｉとｊの最大公約数：　ユークリッドの互除法

(1)　ｉとｊが等しくなるまで繰り返す.

(1-1)　ｉ＜ｊ　ならば，　ｊ　を　ｊ－ｉとする.

さもなければ，　ｉ　を　ｉ－ｊとする.

(2)　ｉまたはｊを最大公約数として返す.

ｉ　＝	３２,	ｊ　＝	１２
	２０		
	８		
			４
	４		

→　最大公約数は　４

再帰関数（繰り返しを再帰で実現）：　gcd（i, j）

i ＝＝　j　であれば，i を返す（再帰の終了条件）．

i ＜ j であれば，gcd（i, j - i）を返す．

さもなければ，gcd（i - j, j）を返す．

```
#include <stdio.h>

#define    M    32
#define    N    12

int gcd (int i, int j)
{
  if (i == j)
    return i;
  else if (i < j)
    return gcd(i, j - i);
  else
    return gcd(i - j, j);
}
```

```
int main(void)
{
  printf
    ("GCD of %d and %d is %d. ¥n",
     M, N, gcd(M, N));
}
```

GCD of 32 and 12 is 4.

素数（prime number）を見つける：

エラトステネスのふるい

２からｎの範囲で，素数を探す（２は素数）．

(1) Ｍ を，２からｎの整数の集合とする.

(2) ｋを２とする.

(3) ｎ／２＜ ｋとなるまで，繰り返す.

(3-1) Ｍの各要素に対して，その要素が
ｋの倍数（１倍を除く）であれば，
その要素をＭから除く.

(3-2) ｋの値を１増やす.

M	2	3	4	5	6	7	8	9	10	11	12	13	14
k = 2													
M	2	3		5		7		9		11		13	
k = 3													
M	2	3		5		7				11		13	
k = 4 ・・・													

初期状態

i	2	3	4	5	6	7	8	9	10	11	12	13	14
m[i]	1	1	1	1	1	1	1	1	1	1	1	1	1

k = 2

i	2	3	4	5	6	7	8	9	10	11	12	13	14
m[i]	1	1	0	1	0	1	0	1	0	1	0	1	0

k = 3

i	2	3	4	5	6	7	8	9	10	11	12	13	14
m[i]	1	1	0	1	0	1	0	0	0	1	0	1	0

k = 4

i	2	3	4	5	6	7	8	9	10	11	12	13	14
m[i]	1	1	0	1	0	1	0	0	0	1	0	1	0

k = 5

i	2	3	4	5	6	7	8	9	10	11	12	13	14
m[i]	1	1	0	1	0	1	0	0	0	1	0	1	0

k = 6

i	2	3	4	5	6	7	8	9	10	11	12	13	14
m[i]	1	1	0	1	0	1	0	0	0	1	0	1	0

k = 7

i	2	3	4	5	6	7	8	9	10	11	12	13	14
m[i]	1	1	0	1	0	1	0	0	0	1	0	1	0

k = 8

i	2	3	4	5	6	7	8	9	10	11	12	13	14
m[i]	1	1	0	1	0	1	0	0	0	1	0	1	0