Programming一再帰アルゴリズムー

do-loopはすべて再帰関数で書き直すことができる．計算速度は一般的に落ちるが，記述が極端に簡素化される場合があるので使えると便利．

和の例

もっとも単純な和の例を示す．以下のような和を求めるプログラムがあるとする．

```
> N:=3;
  total:=0;
  for i from N by -1 to 0 do
    total:=total+i;
  end do:
  total;
```

これを再帰関数で書き直すと以下のようになる．

```
> my_total:=proc(i)
  if i=0 then
    return 0;
  else
    return my_total(i-1)+i;
  end if;
  end proc:
```

再帰関数の特徴は，
1 自分自身を呼び出す．
2 終了条件(ここではi=0)がある．
3 増減する引数がある．

```
> my_total(10);
```

Fibonacci数列

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

とつづく数列をFibonacci数列と呼ぶ．これは

$$a_{i+2} = a_{i+1} + a_i$$

で定義される．for-loopで作ると次のようになる．

```
> fibonacci1:=proc(n_max)
  local f0,f1,f2,i;
  f0:=0;
  f1:=1:
  print(f1);
  for i from 1 to n_max-1 do
    f2:=f1+f0;
    f0:=f1;
    f1:=f2;
    print(f2);
  end do:
  end proc:
```

再帰関数を使うと次のようになる．

```
> fibonacci2:=proc(n)
  if ((n=1) or (n=2)) then
    return 1;
  else
    return fibonacci2(n-1)+fibonacci2(n-2);
  end if;
  end proc;
```

だいぶ簡素化されているのが読み取れよう．

```
> for i from 1 to 10 do
    fibonacci2(i);
  end do;
```

課題 1

素数判定プログラムを簡素化すると以下のようになる．これを再帰関数で書き直せ．

```
> my_isprime1:=proc(n)
  local i;
  for i from 2 to n-1 do
    if (irem(n,i)=0) then
      return false;
    end if;
  end do;
  return true;
  end proc;
```

引数は，(n,i)とするのが自然．

課題2

Mapleでは1000個目の素数は以下の関数で求まる．

```
> ithprime(1000);
```

以下のようにして計算時間を計り，for-loop版とrecursive(再帰)版でどの程度の差となるかを比べよ．

```
> s:=time();
  my_isprime1(7919);
  time()-s;
```