様々な繰り返し処理

for 文、while 文、do 文を用いて様々な繰り返し処理を行う

- 繰り返しの中断・再開
- 数列・積分の数値計算
- 素数・素因数分解
- 最大公約数
- その他

Break 文

for 文、while 文、do 文の繰り返し処理を、途中で中断したい場合がある

break 文は、繰り返し文中で繰り返し本体の実行を終了させる

```
while( ... ){
  ...

  if( ... )
    break;

  ...
}
/* breakにより、ここに実行が移る */
```

途中でループを抜けるのに用いる

ループ中で break 文が実行されると、それを囲む繰り返し文（この例の場合は while 文）が終了。一番内側のループのみが終了する。

for 文、do 文でも同じ。

switch 文でも使用される。

Continue 文

1 文字ずつ読み取り、各アルファベットの出現回数を数えるプログラム

```
int count_a, count_b, ...
int a;
while( (a=getchar()) != EOF ){

  switch( a ){
  case 'a': count_a++; break;
  case 'b': count_b++; break;
  ....
  default: continue;
  }

  count_char++;

  /* continueの実行によりここに処理が移る */
}
```

continue 文は、ループ中からループの先頭へ制御を移動

continue 文の実行により、ループ本体の最後へ処理が移る。つまり、繰り返しを判定する式の評価へ移動。continue 文の後の処理はスキップされる。

このプログラムではアルファベット以外の文字数 count_char を数えていない。

ループ本体中に書かれた continue 文は、それ以降の処理をスキップし、繰り替えし式の評価から繰り返しをやり直す。

```
while( 式 ){
  ...

  if( ... )
    continue;
  ...

  /* continueにより、ここに実行が移る */
}
```

break 文、continue 文を使うことで、柔軟な繰り返し処理（式による繰り返し判定＋例外処理）が可能になる。

goto 文

指定されたラベルに移行(ジャンプ)する。

```
while( 式 ){
  ...

  if( ... )
    goto label;
  ...

}

label:
  printf("ここに処理が移る\n");
```

goto 文の実行により、whileループの外のラベル label へ処理が移る。

goto 文の使用は可能な限り避ける!

break 文、continue 文、goto 文を使うことで、柔軟な繰り返し処理(式による繰り返し判定+例外処理)が可能になる。

様々な繰り返し処理

数列 $\{1, 2, 3, 4, ..., n\}$ の和を求めるプログラム

手順:
1) n の入力 $(n > 1)$
2) for 文を用いて和を計算

```
int i, n, sum=0;

scanf("%d", &n);

for(i=1; i<=n; i++)
  sum += i;

printf("総和は %d \n", sum);
```

参考までに

$\sum_{k=1}^{n} k = \frac{n}{2}(n+1)$ である。

同じ動作をするプログラムを while 文・do 文を使って書いてみる。

円周率の計算

$\int_0^1 \frac{dx}{1+x^2} = \frac{\pi}{4}$ であることを利用して円周率の近似値を求めるプログラム

積分区間 $[0, 1]$ を細かく区切り、短冊の面積を合計することで積分を近似

```
int i, n = 100;
double dx = 1.0/n, sum = 0;

for(i=0; i<100; i++)
    sum += 1.0/(1+ (i*dx)*(i*dx))*dx;

printf("%f\n", 4*sum);
```

区間幅を dx とする

積分は足し算にほかならない

$\tan y = x$ として変数変換。積分区間は $0 \le x \le 1 \to 0 \le y \le \frac{\pi}{4}$ $\frac{dy}{\cos^2 y} = dx$ より

$$\int_0^{\pi/4} \frac{1}{1+\tan^2 y} \frac{dy}{\cos^2 y} = \int_0^{\pi/4} dy = \frac{\pi}{4}$$

素数

1 と自分自身以外では割り切れない自然数を素数という。ただし 1 は素数ではない

2, 3, 5, 7, 11, 13, ...

自然数 a が素数かどうかを判定するアルゴリズム

a が、2 で割り切れない (a%2 != 0)
3 で割り切れない (a%3 != 0)
4 で割り切れない (a%4 != 0)
.....
a–1 で割り切れない (a%(a–1) != 0)

⇒ a は素数である

a–1 までためす必要はないのでこのアルゴリズムは効率悪い

具体例

11 は、2, 3, 4, 5, 6, 7, 8, 9, 10 のいずれでも割り切れないので、素数。
35 は、2, 3, 4 で割り切れないが、5 で割り切れるので素数ではない。

素数判定のプログラム

```
int a, i;

printf(“自然数を入力:”);
scanf(“%d”, &a);

i=2;
while( a%i != 0 ){
  i++;
}

if( i==a )
  printf(“%d は素数です\n”, a);
else
  printf(“%d は素数ではない\n”, a);
```

a が i で割り切れなければ i をインクリメント。
a%a は 0 なので、繰り返しは必ず終了。

繰り返し変数 i が a に達した時のみ a は素数である。

このプログラムでは a を i = 2, 3, 4, ..., a–1, で割っているが、i の範囲は floor(sqrt(a)) までで十分である。floor(double x) は x を越えない整数値を返す関数。math.h で定義されている。

少し効率の良い素数判定

$a = 24$ の素数判定には、24 を $i = 2, 3, 4, 5, ..., 23$ で割る必要はない。
24 の平方根 4.899 (sqrt(a))を越えない整数値の範囲 $i = 2, 3, 4$ で十分。

途中 $i = 2$ で割り切れれば直ちにループを抜けて、無駄な割り算はしない。

```
int i, a = 17;


for(i=2; i*i <= a; i++){
  if( a%i == 0)
    break;
}

if( i*i > a)
  printf("%d は素数\n", a);
```

break 文でループを抜ける

途中で割り切れてループを抜けたなら a は素数ではない

フラグ変数

状態を表す値をとる変数をフラグ変数という。フラグ flag = 旗。

素数判定において、自然数 a が素数であるかどうかを判定するアルゴリズムとして次がある。

1) フラグ変数を int 0 に初期化。
2) 繰り返し処理により、a を $i = 2, 3, 4, ...,$ floor(sqrt(a)) で割り、
   割り切れればフラグ変数を int 1 にする。
3) 繰り返し処理が終了した時点で、フラグ変数が 0 であれば a は素数。

```
int warikireta = 0, i, a=41;

for(i=2; i*i <= a; i++){
  if( a%i == 0){
    warikireta = 1;
    break;
  }
} /* end of for */

if( warikireta == 0)
  printf("%d は素数\n", a);
```

フラグ変数を 0 (False) として初期化

途中で割り切れたらフラグ変数を 1 (True) とする。フラグを立ててループを抜ける。

最後にフラグが立っていなければ a は素数。

素因数分解

素因数分解とは、自然数を素数の積の形に分解すること

例：84 = 2*2*3*7

84 を 2 で割ると、42 余り 0（割り切れる）
42 を 2 で割ると、21 余り 0（割り切れる）
21 を 2 で割ると、割り切れない
21 を 3 で割ると、7 余り 0（割り切れる）
7 を 3 で割ると、割り切れない
7 を 4 で割ると、割り切れない
7 を 5 で割ると、割り切れない
7 を 6 で割ると、割り切れない
7 を 7 で割ると、1 余り 0（割り切れる）

素因数分解アルゴリズム

自然数 a に対して、

1) 2 で割り切れるかぎり、a を 2 で割った商を、2 で割ることを繰り返す
2) 3 で割り切れるかぎり、商を 3 で割ることを繰り返す
3) 4 で割り切れるかぎり、商を 4 で割ることを繰り返す
（4 で割り切れるなら 2 で割り切れているので実際は不要）
4) 5 で割り切れるかぎり、商を 5 で割ることを繰り返す

…

5) 割り算の結果が商 1 余り 0 となれば終了

手順 1) は次のように書ける

```
int a, i;

i=2;
while( a%i == 0){
  a=a/i;
}
```

2 で割り切れるかぎり商を 2 で割り続ける

整数同士の除算（割り算）の結果は整数である!

最大公約数

2 つの自然数 a と b の最大公約数を求めたい

例）36 と 42 の公約数は次のようにして求められる。

```
36%2 == 0 && 42%2 == 0 なので 2 は公約数
36%3 == 0 && 42%3 == 0 なので 3 は公約数
36%4 == 0 && 42%4 != 0 なので 4 は公約数ではない
36%5 != 0 && 42%5 != 0 なので 5 は公約数ではない
36%6 == 0 && 42%6 == 0 なので 6 は公約数（これが最大）
36%7 != 0 && 42%7 == 0 なので 7 は公約数ではない

…

36%36 == 0 && 42%36 == 0 なので 36 は公約数ではない
```

36 までためす必要はない
（効率の悪いアルゴリズム）

最大公約数を求める方法(効率悪い)

```
int a, b, i, gcd=1;

scanf("%d %d", &a, &b);
/* a >= b であるとする */

for(i=2; i<=b; i++){
  if( a%i==0 && b%i==0 )
    gcd=i;
}
printf("%d と %d の最大公約数は %d\n", a, b, gcd);
```

新しい公約数 i で gcd の値を更新

少なくとも b 回の割り算が必要。より少ない計算量で最大公約数を求めるアルゴリズムが知られている。

ユークリッドの互除法

最大公約数を求める効率的な方法の一つ。Euclid : 古代ギリシャの数学者

2 つの自然数 a, b の大きい方を x, 小さい方を y とする。

1) x を y で割った余りを r とする。

2) r が 0 でないかぎり、以下 a), b), c) を繰り返す。

a) y を x に代入。

b) r を y に代入。

c) x を y で割った余りを r とする。

3) y が a と b の最大公約数である。

例)$a = 42, b = 24$ の時($x = 42, y = 24$)

42 / 24 は、1 余り 18 ($r = 42\%24$), r の値は 0 ではない。

x=24, y=18 として、24 / 18 は、1 余り 6。r の値は 6(0 ではない)。

x=18, y=6 として、18 / 6 は、3 余り 0。(r の値は 0 となって割り切れる)

6 が最大公約数である。

問題 1 円周率

$$\frac{\pi}{4} = 1 - 2\sum_{n=1}^{\infty} \frac{1}{(4n-1)(4n+1)}$$ であることが知られている。

上式を利用して円周率の近似値を求めるプログラムを作れ

無限級数の和は実際計算不可能だが、十分大きな n (int) の入力により
円周率の近似値が求められる(と期待される)。

```
% ./a.out
級数の和をいくつまで求めますか：1000
円周率の近似値は 3.1415.....
%
```

この色はプログラムによる出力

整数値 int を実数値 double にキャストして割り算を行うこと!

問題 2 素数判定

入力した正の整数値が素数であるかどうかを判定するプログラム。Ctrl-D が入力されるまで判定を繰り返す。

```
% ./a.out
自然数を入力：35
35 は素数ではない。
自然数を入力：9
9 は素数ではない。
自然数を入力：17
17 は素数である。
自然数を入力：-9
入力エラーです。
自然数を入力：Ctrl-D
プログラムを終了します。
%
```

この色はプログラムによる出力

問題 3 素数列挙

100 以下の素数をすべて列挙するプログラムを作れ

入力された自然数 a が素数であるかどうかの判定を行うプログラムに手を加えれば良い。

```
for(a=2; a<=100; a++){

   a が素数であるかどうかの判定

}
```

```
% ./a.out
2
3
5
7
11
13
17
...
```

このアルゴリズムは効率が悪い（計算量が多い）

エラトステネスのふるい、というアルゴリズムが有名である。
配列のところでやる。

問題 4 素因数分解

自然数を入力し、素因数に分解するプログラムを作れ。
エラー処理も行うこと。

```
% ./a.out
自然数を入力：24
24 = 2*2*2*3
% ./a.out
自然数を入力：144
144 = 2*2*2*2*3*3
% ./a.out
自然数を入力：-9
入力エラー
%
```

この色はプログラムによる出力

問題 5 最大公約数

ユークリッドの互除法を用いて、入力した 2 つの自然数の最大公約数を求めるプログラムを作れ。

```
% ./a.out
自然数を2つ入力:54 144
144 と 54 の最大公約数は 18 です。
%
```

この色はプログラムによる出力

問題 6 完全数

完全数とは、約数（自分自身は除く）の和が自身と等しい自然数である。

例）6 の約数は 1, 2, 3 であり、1 + 2 + 3 == 6 であるので、6 は完全数。

6, 28, 496, 8128 は完全数である。

可能な限りたくさんの完全数を探すプログラムを作れ。

```
% ./a.out
見つけた!6
見つけた!28
見つけた!496
見つけた!8128
...
%
```

ヒント：a が完全数かどうかを判定する部分を作成。
これを a に関するループで囲めば良い。

プログラム実行結果の表示

問題 7 友愛数

2 つの自然数について、片方の約数（自分自身は除く）の和が、他方の約数（同じく自分自身は除く）の和に等しくなるとき、これら 2 つの自然数は友愛数の関係にあるという。

例）220 と 284 は友愛数である。

220 の約数：1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110 => 和は 284
284 の約数：1, 2, 3, 71, 142 => 和は 220

可能な限りたくさんの友愛数を探すプログラムを作れ。

```
% ./a.out
見つけた！6, 6
見つけた！28, 28
見つけた！284, 220
見つけた！220, 284
...
%
```

プログラム実行結果の表示

補足 エラトステネスのふるい

N 以下の自然数の中から素数を求めるアルゴリズムに、
エラトステネスのふるいがある。Eratosthenes：古代ギリシャの数学者

ふるい（篩）：粉または粒状のものをその大きさによって選り分ける道具 [広辞苑第五版図版付き]

1) 2 ~ *N* までの整数を用意する。
2) 最小の素数 (2) の倍数をふるいから取り除く。
3) 残っている整数の最小値を新たな素数とする。
4) 最小の素数 (3) の倍数をふるいから取り除く。
5) 以上を繰り返す。

配列の学習でエラトステネスのふるいをプログラムする。ウェブ検索して予め予習しておくこと。