繰り返し処理 while 文 do 文

所定回反復（特定回数の繰り返し）には for 文を用いた

ある手順を、例えば10 回、繰り返す、といった繰り返し処理

問題を 10 題解け、といった繰り返し。

繰り返し回数が明示的に決まらない場合には while 文、do 文を用いる

ある条件が満たされている限り繰り返す、といった繰り返し処理では
繰り返し回数は決まらない。不定回反復。

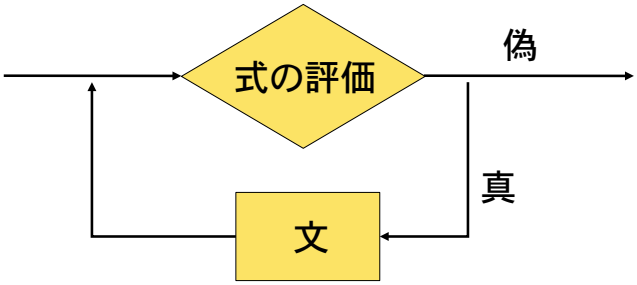
例えば、理解できるまで問題を繰り返し解け、といった繰り返し。

`while(式)文`　式が真であるかぎり、文を繰り返し実行する

繰り返す回数が不定の場合に用いる

while 文

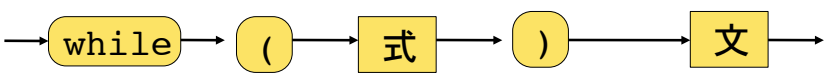
`while(式)文`　式が真であるかぎり、文を繰り返し実行する



文の実行の前に式の評価を行う
前判断反復

一度も文が実行されない場合がある

while 文の構文図



例 1

```
int n;

for(n=0; n<10; n++)
  printf("%d\n",n);
```

繰り返す回数が決まっている場合は for 文を使う場合が多い。

同じ繰り返しを while 文で書いた例

```
int n;

n=0;
while(n<10){
  printf("%d\n",n);
  n++;
}
```

繰り返し変数 n を用意

n を初期化

n の値が 10 未満であるかぎり文（複文）を繰り返す

n の値をインクリメント

n の値をインクリメントしないと無限ループ

正しい繰り返し処理はプログラマの責任

例 2

for 文は単純な置き換えにより while 文に書き直すことができる

```
for(i=10; i>0; i--){
  printf("Count down %d\n", i);
}
```

繰り返し変数 i を 10 に初期化。i > 0 であるかぎり文を繰り返す。文の繰り返し後に i をデクリメント。

```
i=10;
while( i>0 ){
  printf("Count down %d\n", i);
  i--;
}
```

繰り返し変数 i の初期化

繰り返しの条件は i > 0

繰り返し後に i をデクリメント

for 文を while 文に書き直すのは容易（機械的な置き換えで可能）。
逆は必ずしも容易ではない。

繰り返しの終了 scanf

整数の入力を、負の値が入力されるまで繰り返す。何回繰り返すか不定なので while 文を用いる。方法1

```
int data;

printf("整数値の入力(負の値で入力終了):");
scanf("%d", &data);

while( data >= 0){
  printf("data = %d\n",data);
  printf("整数値の入力(負の値で入力終了):");
  scanf("%d", &data);
}
```

ループに入る前に値を読み込んでおく

ループの中で再入力

data の値が零以上であるかぎり、ブロック {...} を繰り返す

ブロック中の scanf 文がない(data の値が更新されない)と無限ループ

データ入力終了のための特殊文字 scanf

scanf() は、特殊文字 Ctrl + D が入力されると、データ入力の終了を意味する EOF という特殊な値(終わりの合図)を返す。(int -1)

特殊文字で繰り返しを終了する場合、入力したデータ値(変数に格納した値)で繰り返しの判定をすることは出来ない。

Ctrl+D が入力されるまで、データ入力を繰り返す常套手段

```
int data;

while( scanf("%d", &data) != EOF ){
  printf("data = %d\n",data);
}
```

while 文の式として　　scanf("%d", &data) != EOF　　を指定

⇩

scanf によるデータ入力。通常の入力では変数 data に値が格納される。入力が Ctrl+Dの場合は scanf() 自体が EOF を返す。

例 3

Ctrl+D が入力されるまで整数値を繰り返し読み込み、読み込んだデータの個数を表示するプログラム

何個のデータを読み込むか不定なので while 文を用いた繰り返し処理になる

```
int data, count=0;

while( scanf("%d", &data) != EOF ){
  printf("data = %d\n",data);
  count++;
}

printf("データの個数は %d\n",count);
```

入力回数を数える変数を宣言と同時に初期化

データ表示するごとに変数 count をインクリメント

特定文字の入力で繰り返しを終了 getchar

特定文字（例えばピリオド）が入力されるまで getchar() で文字を繰り返し読み込む

変数に文字を格納し、変数値が特定文字かどうかを繰り返しの条件にすれば良い

```
int code;

code = getchar();
while( code != '.' ){
  printf("文字 %c = 文字コード %d\n", code, code);
  code = getchar();
}
```

入力文字が '.' で無いかぎりブロックの実行を繰り返す例。

ブロック中で code の更新（再入力 code = getchar() ;）が無いと、無限ループ。

データ入力終了のための特殊文字 getchar

getchar() は、特殊文字 Ctrl+D の入力があると EOF という値を返す。

```
int code;

while( (code=getchar()) != EOF ){
  printf("文字 %c = 文字コード %d\n",code,code);
}
```

式 (code=getchar()) != EOF は、

getchar() で読み込んだ 1 文字を変数 code に代入し、その値が EOF でなければ真、そうでなければ偽、となる式を表す。代入式は値を持つ。

変数 code に getchar() の返却値を代入した後、code != EOF を判定する手順を C 言語では、(code = getchar()) != EOF と書くことができる。

カッコ ( ) が必要である。カッコがないと、先に getchar() != EOF が評価され、その結果が変数 code に代入される(!= の方が = よりも優先順位が高いため)。
プログラムの動作は全く異なってくる。

入力のバッファリング

scanf() や getchar() を用いてデータの入力をするとき、実際の入力は改行文字(リターン)が入力されて初めて開始される。こうした行単位による処理をバッファリングという。

キーボードから入力したデータは、プログラムにすぐに受け渡されるのではなく、入力バッファと呼ばれる一時的な記憶領域に格納され、改行文字の入力、もしくは所定量のデータ入力(バッファが一杯になる)によって初めてプログラムに受け渡される(バッファのフラッシュ)。

処理系によって、バッファリングの処理(改行文字がプログラムに受け渡されるかどうか)が異なるので、意に反したおかしな動作をする場合がある。

入力データはバッファにたまっていく。

改行文字の入力により入力データがプログラムに受け渡される。

バッファリングの例

```
int code;

while( (code=getchar()) != EOF ){
  printf("文字 %c = 文字コード %d\n", code, code);
}
```

上記プログラムの実行例

```
%./a.out
abc
文字 a = 文字コード 97
文字 b = 文字コード 98
文字 c = 文字コード 99
文字
 = 文字コード 10
```

このプログラムを実行すると、入力した文字以外に、文字コード 10（改行文字）が表示される（バッファリングのため）

改行文字 '\n' が入力されて初めて、a, b, c, '\n' の 4 文字がプログラムに受け渡される。

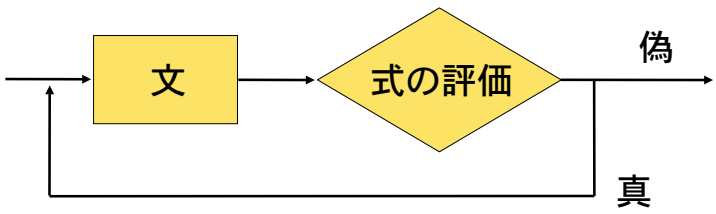
改行文字もプログラムに受け渡されることに注意。

do 文

不定回繰り返しで、繰り返しの条件を文の実行後に判定する後判定反復

```
do 文 while(式)
```

文の実行後、式の評価が真であれば文を繰り返す。偽であれば do 文の終了。

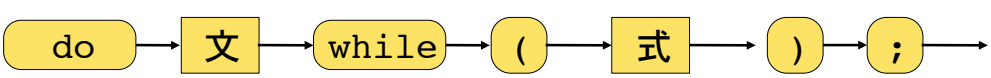


文の実行後に、繰り返しの判定を行うのが do 文。

文は最低 1 回は実行される。

while 文は 1 回も文が実行されないことがある。

do 文の構文図



do 文の例

```
i=0;
do{
  printf(“%d\n”,i);
  i++;
}while(i<10);
```

for 文に書き直すと

```
for(i=0; i<10; i++)
  printf(“%d\n”,i);
```

入力値が正であるかぎり入力を繰り返す

```
int data;
do{
  printf(“整数を入力(負値で終了): ”);
  scanf(“%d”, &data);
}while(data>=0);
```

同じことは while 文を使っても可能

補足 1

特定の値の入力(この場合は int 0)で繰り返しを中断する例

```
int data;

printf(“data = %d\n”,data);
scanf("%d", &data);

while( data != 0){
  printf(“data = %d\n”,data);
  printf(“整数値の入力(0 で入力終了):”);
  scanf(“%d”, &data);
}
```

do 文を使う例。

```
int data;

do{
  printf(“data = %d\n”,data);
  scanf("%d", &data);
} while( data != 0)
```

正しい入力が行われるまで繰り返す処理例

```
int x, y, dummy;

while( scanf("%d %d", &x, &y)!=2 ){
  printf(“入力エラーです\n”);
  while( (dummy=getchar())!='\n' );
}
```

←

バッファリングにより繰り返し処理の判定式 scanf("%d %d", &x, &y)!=2 が正しく判定されない。
ダミー変数 dummy を用いて、改行文字の入力があるまで入力データを読み飛ばす。この while 文が無いとどうなるか確認してみよ。

補足 2

特定文字の入力で繰り返しを中断する例（過ちである。うまく動かない）

```
int code;

while( code = getchar() != EOF ){
  printf("文字 %c = 文字コード %d\n",code,code);
}
```

繰り返しの判定式　`code = getchar() != EOF`　は、!= の方が = よりも優先されるので

getchar() != EOF がまず最初に評価される。入力値が Ctrl+D でないと、この式は真 (int 1) である。
その後、int 1 が変数 code に代入される。

正しい繰り返しの判定式 `(code=getchar()) != EOF` とはまったく異なる結果になる。

上のプログラムは構文的には正しいのでコンパイル可能。しかし、正しく動作しない。
正しい繰り返し処理はプログラマの責任である。

代入演算子再考

Ctrl-D が入力されるまで 1 文字ずつ読み込むループの例

```
int code;

while( (code = getchar()) != EOF ){
   ....
}
```

代入演算子 = は右辺の式の値を左辺の変数に代入する。

例 x = 1　　これを代入式と呼ぶ。代入式自身も値を持つ。その値は代入された値に等しい。

```
int x=5;

printf("%d", x);
printf("%d", x=5);
```

printf("%d", x); ←―― 変数 x の値を表示
printf("%d", x=5); ←―― 代入式 x=5 の値を表示

どちらの表示も 5 となる。

代入演算子の連接

a = 1　　変数 a に 1 を代入

a = b = c = 1　　と書くと、変数 a, b, c に 1 が代入される（代入演算子の連接）

その仕組みは、次の通り。

変数 a に代入式 b=c=1 の値を代入する。

a = (b = c = 1)

代入式 b=c=1 の値は、変数 b に代入式 c=1 の値、つまり 1 を代入したものである。

a = (b = (c = 1))

以上の結果、変数 a, b, c に 1 が代入される。

式の値補足

if 文の式として、いろいろな演算子を用いた式を学んだ。

if(x > 0) ... , if(a == b) ... , if (a < 0 && b !=0) ..., などなど

これらの式の値は、条件が成り立てば 1 (int), そうでなければ 0 (int) となる。

```
int a = 3;
double x = 3.1415;

printf("%d\n", a == 2);
printf("%d\n", x > 0 );
```

←―― a == 2 は偽なので 0 と表示

←―― x > 0 は真なので 1 と表示

```
if( 3 < x < 5 ) ...
```

と数学風に書いてはいけない理由
（構文的には正しいが正しく動作しない）

式 3 < x < 5 は、(3 < x) < 5 と解釈されるので x の値に関わらず常に真 1 となる

問題 1

先週作成した九九の表を for 文ではなく、while 文を使って作れ。

ヒント：while 文の入れ子になる。繰り返し変数の初期化に注意！

```
% ./a.out
1*1 =  1, 1*2 =  2, 1*3 =  3, ... 1*9 =  9
2*1 =  2, 2*2 =  4, 2*3 =  6, ... 2*9 = 18
3*1 =  3, 3*2 =  6, 3*3 =  9, ... 3*9 = 27
...

9*1 =  9, 9*2 = 18, 9*3 = 27, ... 9*9 = 81
%
```

この色はプログラムによる出力

for 文を while 文に変換するのは機械的な置き換えで可能である。

問題 2

キーボードから整数値を読み込む。Ctrl+D の入力でデータ入力を終えた後、読み込んだ整数値の合計を表示するプログラム。

ヒント：データを何個読み込むか不定なので while 文による繰り返しとなる

```
% ./a.out
整数を入力：10
整数を入力：20
整数を入力：30
整数を入力：40
整数を入力：Ctrl-D（実際には表示されない）
入力したデータは 4 個、総計は 100 です。
%
```

この色はプログラムによる出力

問題 3

改行文字が入力されるまで文字を読み込み、入力した文字の数を数えるプログラムを作れ。getchar() を使うこと。
ただし、空白文字（スペース）や記号なども 1 文字と数える。

```
% ./a.out
文字を入力：abcdefg
文字数は 7 文字です。
% ./a.out
文字を入力：How are you?
文字数は 12 文字です。
%
```

ヒント：読み込んだ文字が改行文字 '\n' であれば繰り返しを終了する。

言い換えると、読み込んだ文字が改行文字 '\n' でないかぎり繰り返しを継続。

文字数をカウントするには、該当する文字の入力があった時に、文字数をカウントする変数値をインクリメントすればよい。

問題 4

Ctrl-Dが入力されるまで英文（改行文字を含む）を読み込み、入力した文章の行数、単語数、および文字数（記号を含む）を表示するプログラム。

行数は入力された改行文字、単語数は空白文字（スペース）を数えればわかる。

```
% ./a.out
文章を入力：
Hello!
How are you? [Ctrl-D]
文章は 2 行、単語は 4 つ、文字は 16 文字です。
%
```

ヒント：getchar() で一文字ずつ読み込む。読み込んだ文字が、改行文字、空白文字であるかを判定して、行数、単語数を数えればよい。

問題 5

数列 $a_n = n^2,\ \{1, 4, 9, 16, ..., k^2\}$ の和が 10000 を越える k を求めよ。

ヒント：数列 a_n の和が 10000 以内であるかぎり足し続ける。

問題 6

ある塩基配列を読み込み、A, T, G, Cの出現回数を数えるプログラム

Mathematica の データベースを用いてある遺伝子の塩基配列を読み込み、
これをテキストファイルとして書き出す。

getchar() を用いて、このテキストファイルに書かれている内容を読み込み、
各延期の出現頻度を数えるプログラムである。

UNIX の知識

端末エミュレータではシェル shell と呼ばれるプログラムが動作している。
シェルはユーザが入力するコマンドを実行する。

UNIX では、通常の入力（標準入力）は、キーボード、出力（標準出力）はモニターディスプレイ、に設定されている。（だから入力コマンド・データはキーボードから入力し、その結果はモニターに表示される）

シェルが持つ機能の 1 つにリダイレクション redirection がある。リダイレクションとは入力元や出力先を変更する機能。< と > を用いる。

% command < file_in	コマンド command への入力をキーボードではなく、file_in というファイルに指定
% command > file_out	コマンド command の出力をモニターではなく、file_out というファイルに指定

リダイレクションの応用

while 文を用いて入力文字数・単語数を数えるプログラムを作成した（右）

```
% ./a.out
How are you?
Ctrl-D
3 words, 12 characters including space.
%
```

予め、入力する文章をテキスト形式のファイルに用意しておくと、リダイレクションにより、入力元をこのファイルに指定することができる。

```
% ./a.out < shakespeare.txt
12345 words, 98765 characters including space.
%
```

ファイルの最後には EOF (End Of File) が書き込まれているので、標準入力にて Ctrl-D を入力するのと同じ仕掛けで読み込みループが終了する。

この例では入力元をファイルへリダイレクトしているが、出力先は標準出力（モニター）のままなので、プログラムの実行結果はモニターに表示される。

```
% ./a.out < shakespeare.txt
12345 words, 98765 characters including space.
%
```

出力先をファイルに指定すると、新規にファイルが作られ、その中身はプログラムの動作結果が書き込まれている。

```
% ./a.out < shakespeare.txt > result
% cat result
12345 words, 98765 characters including space.
%
```