プログラミングI
数理物理，総合理学等向け

2018年10月29日

海谷 治彦

目次

- 演習5の解答例

- while文による繰り返し
- i++ 等の略記法
- 段付け(indent)をしよう.
- break文
- continue 文

- 演習5

前回の

本日の演習5

- 身長と体重を入力し，身長から体重を引いた値が，
  - 100以下の場合，「やせろ」
  - 120以上の場合，「ふとれ」
  - それ以外は無言

と生意気に助言するプログラムを作成せよ.

期待される結果の例

```
$ cc -o sbmi2 sbmi2.c
$ ./sbmi2
175
80
height = 175 weight = 80
Lose your weight!
$
```

コレはユーザーが
175と80を入力して，
それぞれエンターを押す
ものとしてください.

TIPS ファイル名について

- nantoka.c 等のファイル名は以下にしてください.

1. 日本語や倍角文字は使わない.
   × なんとかA.c
2. 空白文字を含めない.
   × program 1a.c
3. 小文字の英数文字で構成する.
   ○ program1a.c
4. 特殊記号は原則使わない. _ と – はOK.
   × %!my/prog.c
   ○ my_prog.c

- OSにおけるファイル名の規則は上記より緩いのですが, プログラムに関わるファイル名では, 上記にしてください.

プログラムの基本動作

- プログラムは命令文のリストで構成されてます.
- コンピュータは命令文を書いてある順番に順々に実行します. (原則)
- しかし, それだけでは込み入った処理がかけないので, 以下の三種類があります.
  1. 条件分岐: 条件によって特定の命令を実行せずに飛ばす. (if文)
  2. 繰り返し: 前に実行した命令に戻って再度実行する.
  3. 関数呼び出し: 予めグループ化した命令群を繰り返し利用する. (printf や gets 等)
- 本日は二番目の繰り返しについて学びます.

p.80-
(明)

命令群の繰り返し

- 繰り返す内容
  - 全く同じ繰り返すのは稀.
  - 通常は一部異なるが似たような処理を繰り返す.
    - list6-1x, list6-2xはprintする数が異なる
  - ほとんどの場合，命令は同じだが，その実行に使うデータが異なるという繰り返しが多い.
    - 命令はprintfだが，表示する変数が異なる
    - 命令は足し算だが，付け足す数が異なる
- 繰り返す回数や内容
  - 「ある条件が成り立つ限り繰り返す」場合: 今回話すwhileを使うのが一般的.
  - 「N回繰り返す」と数を決めてしまう場合や物の集まり(集合やリスト)への同じ処理の適用: while でも書けるが，より適した書き方もある (8回目に紹介)

while文

- 繰り返しを行う構文.
- 構文の意味は以下,

  「ある条件が成り立っている限り，ブロック内の命令群を繰り返す」

- ブロック内の命令によって，変数等の値が変わることで，繰り返す毎に条件判断が変わることが多い.
- 構文の形は if 文に似ている.
  - if が while になっただけ

```
while(条件){
   繰り返す命令群;
}
```

whileの特長

- if文を繰り返し判断する感じの意味.
- 構造も意味もif文に似ている.
  - if文 ⇒ 条件が成り立てば，指定処理を実行.
  - while文 ⇒ 条件が成り立つ限り，指定処理を繰り返し実行.

- 無限ループも書ける.
  - while(1) と通常かく.
- 無限ループの場合，本当に無限に続く分けでなく，途中でループを抜ける命令がある.
  - break 文.

whileを使う場合/使わない場合

- ここでは3回繰り返すという単純な例.
- 3回なら list6-1x のようにかけるが，100回だったら，while無しにプログラムを書いてられない！

```
// list6-1x.c
#include <stdio.h>

int main(void){
  printf("%d¥n", 0);
  printf("%d¥n", 1);
  printf("%d¥n", 2);
  return 0;
}
```

```
bash-3.1$ cc list6-2x.c
bash-3.1$ ./a.exe
0
1
2
bash-3.1$
```

同じ意味

```
// list6-2x.c
#include <stdio.h>

int main(void){
  int i=0;
  while(i<3){
    printf("%d¥n", i);
    i=i+1;
  }
  return 0;
}
```

実行される命令を展開

```
// list6-2x.c
#include <stdio.h>

int main(void){
  int i=0;
  while(i<3){
    printf("%d¥n", i);
    i=i+1;
  }
  return 0;
}
```

```
  i=0;
  if(i<3){ // 今 i==0
    printf("%d¥n", i);
    i=i+1;
  }
  if(i<3){ // 今 i==1
    printf("%d¥n", i);
    i=i+1;
  }
  if(i<3){ // 今 i==2
    printf("%d¥n", i);
    i=i+1;
  }
  if(i<3){ // 今 i==3
    // 成り立たないのでループ終了
  }
```

代入文の略記法

- 以下の略記がCではよく使われる.
- タイプを短くしたいという要望から生まれたものと推測される.
- * や / や % にも使える.
- x++ と ++x の意味が違ってくる場合があるが，その話は後日.

	本来	略記
1個増やす	x=x+1	x++
1個減らす	y=y-1	y--
N個増やす	a=a+N	a += N
N個減らす	b=b-N	b -= N

変数値の初期化

- C言語では変数の宣言(定義)時点では値は決まっていない.
- しかし，明示的に初期値を設定することもできる.
- 初期値を決めても，その後，値の変更は自由にできる.

```
// list6-2x.c
#include <stdio.h>

int main(void){
  int i=0;
  while(i<3){
    printf("%d¥n", i);
    i++;
  }
  return 0;
}
```

同じ意味

```
// list6-2y.c
#include <stdio.h>

int main(void){
  int i;
  i=0;
  while(i<3){
    printf("%d¥n", i);
    i++;
  }
  return 0;
}
```

N回繰り返すの典型パターン

- while文は多様な繰り返しを記述することができる.
- 単純に「N回繰り返す」を以下の表現で書く場合が非常に多いので，そのパターンを覚えよう.
- 尚，プログラムの世界では伝統的に，N数えるのに，「1, 2, 3, 4 … N」ではなく，「0, 1, 2, 3, …. N-1」とする.

```
int i=0;
  while(i<3){ // コレは 0, 1, 2 と数えている
    printf("%d¥n", i); // print命令は同じ iが毎回違う
    i++;
  }
```

```
int i=1; // 以下は使わないのが普通. 理由もある.
  while(i<=3){
    printf("%d¥n", i);
    i++;
  }
```

List 6-3 改

```
// list6-3x.c 10回繰り返しだけど 0, 1, 2 ... 9 と数える
#include <stdio.h>

int main(void){
  int x=0;
  while(x<10){
    printf("Square of %d: %d, ", x, x*x);
    printf("Cube of %d: %d¥n", x, x*x*x);
    x++;
  }
  return 0;
}
```

```
bash-3.1$ cc list6-3x.c
bash-3.1$ ./a
Square of 0: 0, Cube of 0: 0
Square of 1: 1, Cube of 1: 1
Square of 2: 4, Cube of 2: 8
Square of 3: 9, Cube of 3: 27
Square of 4: 16, Cube of 4: 64
Square of 5: 25, Cube of 5: 125
Square of 6: 36, Cube of 6: 216
Square of 7: 49, Cube of 7: 343
Square of 8: 64, Cube of 8: 512
Square of 9: 81, Cube of 9: 729
bash-3.1$
```

whileやifの入れ子

- whileやif文は，入れ子にして組み合わせて使うことができる.
- 一般的に二次元的なデータは二重のwhile入れ子，三次元的なデータは三重の入れ子で処理できる.

- 次のページの例では，二次元的な棒グラフを描いてみている.

List 6-4 改

```
// list6-4x.c
#include <stdio.h>

int main(void){
  int i=0; // 行を数える
  while(i<10){ // 行を書くループ
  int j=0; // *の数を数える
    printf("%d ", i);
    while(j<i){ // *を書くループ
      printf("*");
      j++;
    }
    printf("¥n");
    i++;
  }
  return 0;
}
```

```
bash-3.1$ cc list6-4x.c
bash-3.1$ ./a
0
1 *
2 **
3 ***
4 ****
5 *****
6 ******
7 *******
8 ********
9 *********
bash-3.1$
```

段付け(indent)をしよう

- ブロックの開始と終了，if や while の入れ子の見た目が分かりやすいように，プログラムの段付けをしてください！
- 右の例を参照.
- 基本，{} があれば，その内側は一段下げる.
  - 空白文字かタブ
- 多くの受講生は段付けをしてないので，プログラムが見難くない？

```
// list6-4x.c
#include <stdio.h>

int main(void){
  int i=0; // 行を数える
  while(i<10){ // 行を書くループ
  int j=0; // *の数を数える
    printf("%d ", i);
    while(j<i){ // *を書くループ
      printf("*");
      j++;
    }
    printf("¥n");
    i++;
  }
  return 0;
}
```

N回繰り返しじゃない例

- ユークリッド互除法による最大公約数(GCD)の計算.
  1. 相互に同じ数なら，その数がGCD.
  2. そうでなければ，大きい値から小さい値を引く.
     - 本来，互除法では割った余りを使うが，相互に引いていっても結果は同じ．処理効率が違う.
  3. 再度，1に戻るという繰り返し.
- 同じ処理の繰り返しで意味ある計算ができる典型的な例であり，コンピュータのありがたみを典型的に表している.

```
// gcd.c 最大公約数を求める
#include <stdio.h>

int main(void){
int a, b;
  // 最大公約数を求めるための正整数を2個入力してもらう
  scanf("%d", &a);
  scanf("%d", &b);
  if(a<1 || b<1) { // どっちか負や0なら警告出して終了.
    printf(“Positive integers expected! %d %d¥n", a, b);
    return 1;
  }
  printf("Gcd of %d and %d is ", a, b);
  while(a!=b){
    if(a>b){
      a = a-b; // a -= b; と書いてもよい
    }else{
      b = b-a; // b -= a; と書いても良い
    }
  }
  printf("%d. ¥n", a);
  return 0;
}
```

```
bash-3.1$ ./a
12
18
Gcd of 12 and 18 is 6.
bash-3.1$ ./a
33
18
Gcd of 33 and 18 is 3.
bash-3.1$
```

繰り返し条件判定は最初に

- while文では，繰り返しの条件判定は，繰り返す前に行なわれる.
- よって，「なんらかの処理」(ユーザー入力等を含む)の結果によって，繰り返すか否かを判定する処理は書きにくい.

- 次頁の例では，ユーザーがゼロを入力するまで数値を受け取り，その合計値を計算している.
- ユーザー入力の処理(一回目用)をwhileの外に書かないといけないので，ちょっと冗長である.

```
// inputsum1.c
#include <stdio.h>

int main(void){
int sum=0, val;
  scanf(“%d”, &val); // 1回目の入力のためだけに使われる
  while(val>0){
    sum += val;
    scanf(“%d”, &val); // 3行上と同じことが書いてある
  }
  printf("sum = %d¥n", sum);
  return 0;
}
```

```
bash-3.1$ cc inputsum1.c
bash-3.1$ ./a
1
2
3
4
0
sum = 10
bash-3.1$ ./a
0
sum = 0
bash-3.1$
```

p.97-
(明)

break
ループを途中で抜ける

- 繰り返しを途中で抜けることができる.
- break という命令を使う.
- 通常，if文と組み合わせて，特定条件下で抜け出す等を書く.
- break ではなく return 文で関数そのものから抜ける手もあるが，それは後日に.

前の例が少しマシになる

- whileの繰り返し条件が 1 すなわち無限ループとなっている.
- 途中のbreakが無ければループから抜け出せない.

```
// inputsum2.c
#include <stdio.h>

int main(void){
int sum=0, val;
  while(1){
    scanf(“%d”, &val); // scanfはここ一箇所
    if(val<=0) {
      break;
    }
    sum += val;
  }
  printf("sum = %d¥n", sum);
  return 0;
}
```

```
bash-3.1$ cc inputsum2.c
bash-3.1$ ./a
0
sum = 0
bash-3.1$ ./a
4
1
2
3
0
sum = 10
bash-3.1$
```

p.101-
(明)

continue
命令群を中断して次の繰り返しをする

- 繰り返すブロック内の命令のリスト途中で，残りの命令を実行せず，次の繰り返しを行うことができる.
- if文で以降のブロックをくくってもよいが，それよりもシンプルにプログラムがかける.

例 割り切る数の表示

```
// div357a.c 3, 5, 7 で割れる数は飛ばす
#include <stdio.h>

int main(void){
int i=0;
  while(i<30){
    i++;
    if(i%3==0) continue;
    if(i%5==0) continue;
    if(i%7==0) continue;
    printf("%d ", i);
  }
  printf("¥n");
  return 0;
}
```

```
sh-3.1$ ./a.exe
1 2 4 8 11 13 16 17 19 22 23 26 29
sh-3.1$
```

```
// div357b.c
// 左と同じ意味
#include <stdio.h>

int main(void){
int i=0;
  while(i<30){
    i++;
    if(!(i%3==0 || i%5==0 || i%7==0)) {
      printf("%d ", i);
    }
  }
  printf("¥n");
  return 0;
}
```

逆順での繰り返し

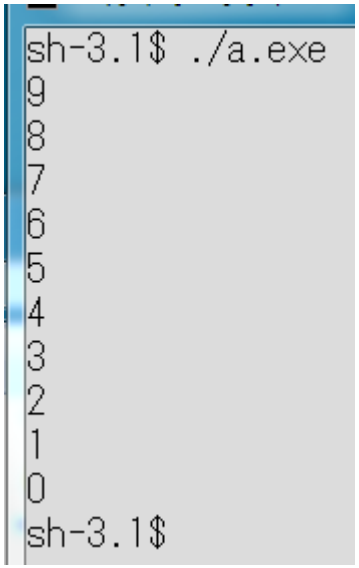
- N回繰り返しは逆順で行ってもよい.
- 場合によっては，逆順のほうが便利なことがあるかも.

```
// revcount1.c
#include <stdio.h>

int main(void){
int i=10;
  while(i>0){
    i--;
    printf("%d¥n", i);
  }
  return 0;
}
```

```
// revcount2.c
#include <stdio.h>

int main(void){ // 表示だけ逆順
int i=0;
  while(i<10){
    i++;
    printf("%d¥n", 10-i);
  }
  return 0;
}
```



本日の演習6

- 0より大きい整数値 x を入力すると，0以上x以下の奇数の値全てを表示するプログラムを作成せよ.

  ヒント 奇数とは2で割った余りがゼロで無い数である.

- odd.c というファイル名で提出せよ.

期待される結果の例

5行に分けて，
書いてもOK

入力	結果
10	1, 3, 5, 7, 9
5	1, 3, 5
2	1
0	無言，もしくはエラー警告

以上