プログラミングI
数理物理，総合理学等向け

2018年12月17日

海谷 治彦

目次

- 教科書[レ] 8章 関数について
- なぜ，関数が必要か？
- 関数の使い方
- 関数の構造
- 関数の作り方

- 前回の演習 解答例

- 本日の演習 (演習11)

プログラム作りへの要求

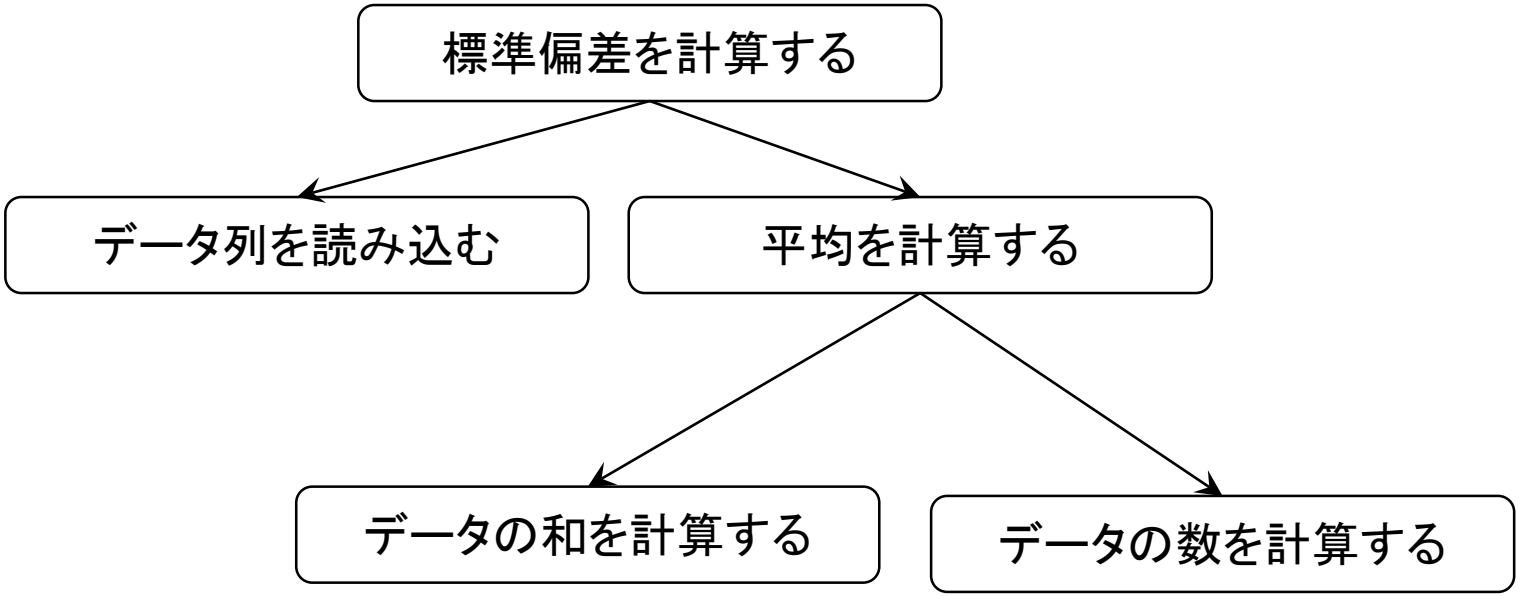
1. 同じような処理内容を，毎回，毎回，書くのはイヤだ．簡易に使いまわしたい．
2. 他人の作った処理は，そのままパクりたい．コピペさえ面倒．
3. 手分けしてプログラムを作りたい．

- 関数というのは，これらの要求を満たす！

なぜ，関数が必要か？

- C言語に限らず複雑な処理(機能)は，より単純な機能の組み合わせとして表現することで，人間の手におえる規模になります．
  - 構造化とか詳細化とか呼ばれる機能分解です．
- 関数は，この複雑な機能を単純な機能に分解するための，典型的な手段です．
- 関数を呼び出す(利用する)時には，
  - 内部でどのように処理するか(how)は気にせず，
  - 何を処理してくれるか(what)がわかっていれば十分です．
    - これを仕様と呼びます．
- この特徴が複雑な機能を組み立てたり，理解したりする助けとなっています．

簡単な機能分解の例



データをX1，X2 … Xn、平均値を$\overline{X}$としたとき

分　散　：　$$S^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{x})^2$$

標準偏差　：　$$S = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{x})^2}$$

復習

参考: 実プログラムの規模

```
   101 undo.h
   603 version.c
   455 window.c
   396 word.c
 32734 total
[kaiya@kaiya2014 ng]% ls *.[ch]
afo.c       def.h       kanji_.c    macro.c     regex_j.c   ttydef.h
basic.c     dir.c       kanji.c     macro.h     regex_j.h   ttyio.c
buffer.c    dired.c     kanji_.h    main.c      region.c    ttykbd.c
canna.c     display.c   kbd.c       match.c     re_search.c undo.c
chrdef.h    echo.c      kbd.h       modes.c     search.c    undo.h
cinfo.c     extend.c    key.h       paragraph.c shell.c     version.c
cmode.c     file.c      keymap.c    random.c    skg.c       window.c
complt.c    fileio.c    kinit.h     regex.c     spawn.c     word.c
complt.h    help.c      kinsoku.c   regex_e.c   sysdef.h
config.h    jump.c      line.c      regex_e.h   tty.c
[kaiya@kaiya2014 ng]% ls *.[ch] |wc
     58      58     536
[kaiya@kaiya2014 ng]%
```

上記は，とても小さなテキストエディタのソースプログラム群だが，
ファイル数で 58個，
総行数は 3万2千行ほどある．(.c .h のみ)

Linuxは中核の部分だけで1000万行を超えているらしい．
Emacs-24.3は38万行以上，ファイルは360個 (.c .h のみ)

関数の効用

- 再利用を促進し，無駄な繰り返しを避けることができる.
- 多くの人に再利用されている関数は大抵，高品質で高性能である.
  - すでに先人が動作テストしているも同然なので.
- 誤りがあった場合，関数自体を直せば，それを使っている部分全部が正常に動くようになる.
  - 同じ処理をあちらこちらに複数コピーを書いた場合，誤りを直すのが大変なことになる.

関数を使う

- 使う関数の仕様を理解する.
  - 名前は何か？
  - どんな機能か？
  - 引数はいくつあってどんな型か？
  - 返り値の型は何か？
- 必要ならコンパイル時に関数が封入されているライブラリを指定する.
  - 標準的なものは無条件に利用可能となっている.
    - atoi, scanf, fputs 等
  - そうでないものはライブラリというのの指定が必要.
    - 次頁の例: 画像処理を行う関数群は追加指定が必要.

復習

strcpy ついでに strlen

- それぞれ文字列に関する操作関数
- #include <string.h> が必要

```
char* strcpy(char* dst, char* src)
```

- srcからdstにコピーする. 方向をすぐに忘れるわorz
- 返値は dst と同じ値を返す.

```
size_t strlen(char* s)
```

- 文字列sの長さを返す, '¥0'は含まない長さ.
- 副作用は無い.
- size_t は当面 int と同じと考えてよい.

復習

文字配列の代入 (コピー)

- 配列は通常変数のように丸ごとコピーできない.
- コピーするには要素を一個一個コピー(代入)しないといえない.
- 文字配列の場合，以下の処理を行うが，同じ処理を行う関数 strcpy が用意されている.

```
// Nは適当に定義されているものとする
// src[]は初期化されているものとする
// src から dstにコピーするものとする
char src[N], dst[N];
int i=0;
  while(src[i]) {dst[i]=src[i]; i++;}
  dst[i]=src[i]; // '¥0' のコピーもする
```

復習 str.c 文字列系関数のサンプル

サンプル

```
#include <stdio.h>
#include <string.h>

int main(void){
char buf[101], dest[101];
  // scanfの実行成功した返値は読み込んだ変数の数
  if(scanf("%100s", buf)!=1) return 1;

  // コピーを行う 引数2から引数1へ
  strcpy(dest, buf);
  // 用心のため元をつぶす
  strcpy(buf, "");

  printf("buf=[%s] dest=[%s] %d¥n",
    buf, dest, strlen(dest));

  return 0;
}
```

```
bash-3.1$ cc str.c
bash-3.1$ ./a
sakana
buf=[] dest=[sakana] 6
bash-3.1$
```

scanf データの入力関数

- scanf(書式, 入力する変数･･･)
- 返り値 入力できたデータの数
- 非常に多機能な関数であり，本授業では，以下の限定的な使い方しか行なわない．
- 整数の入力 変数 `int x` に対して，

```
scanf("%d", &x)
```

- 実数の入力 変数 `double d` に対して，

```
scanf("%lf", &d)
```

- 文字配列の入力 変数 `char str[11]` に対して，

```
scanf("%10s", str) // コレだけ&が不要
```

反例とライブラリの指定

下記のプログラムでは jpeg (画像ファイル)を扱う関数を使ってますが,
ライブラリを指定していないので,「そんな関数は知らん！」というエラーが大量に出ます.

```
[kaiya@oboe31 ~]$ cc jpegSample.c
/tmp/cceskitN.o: In function `main':
jpegSample.c:(.text+0x17): undefined reference to `jpeg_std_error'
jpegSample.c:(.text+0x37): undefined reference to `jpeg_CreateDecompress'
jpegSample.c:(.text+0x66): undefined reference to `jpeg_stdio_src'
jpegSample.c:(.text+0x7a): undefined reference to `jpeg_read_header'
jpegSample.c:(.text+0x89): undefined reference to `jpeg_start_decompress'
jpegSample.c:(.text+0x125): undefined reference to `jpeg_read_scanlines'
jpegSample.c:(.text+0x144): undefined reference to `jpeg_finish_decompress'
jpegSample.c:(.text+0x153): undefined reference to `jpeg_destroy_decompress'
collect2: ld はステータス 1 で終了しました
[kaiya@oboe31 ~]$
```

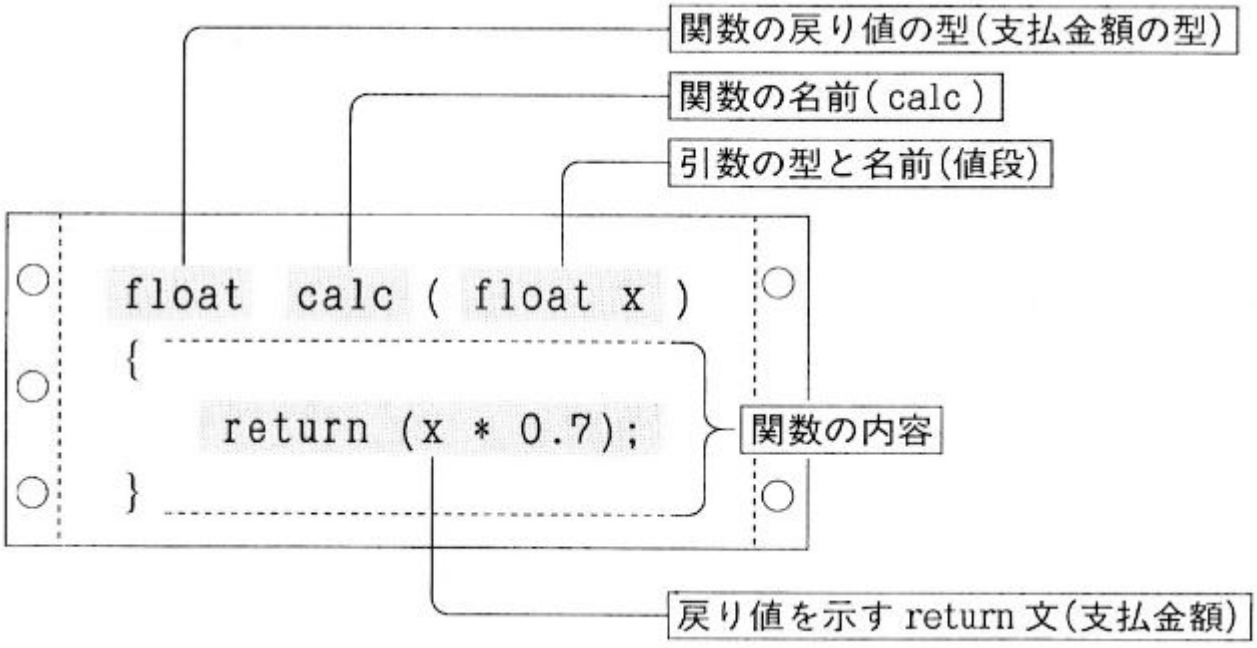
これら関数が入っているライブラリを指定すると, エラーもでませんし, ちゃんと動作します.

```
[kaiya@oboe31 ~]$ cc jpegSample.c -ljpeg
[kaiya@oboe31 ~]$
```

jpegのデータ解析プログラムを全て手作りで作ることは可能ですが, 大変な量になります.
ちなみに, 上記プログラムは80行程度で, jpegをhtmlに変換します.
http://www.syuhitu.org/other/jpeg/jpeg.html

関数の構造 Fig. 8-2 より

- 名前，返り値の型，引数の名前と型(0～複数個)
- プロトタイプ宣言もこれらが指定されている.

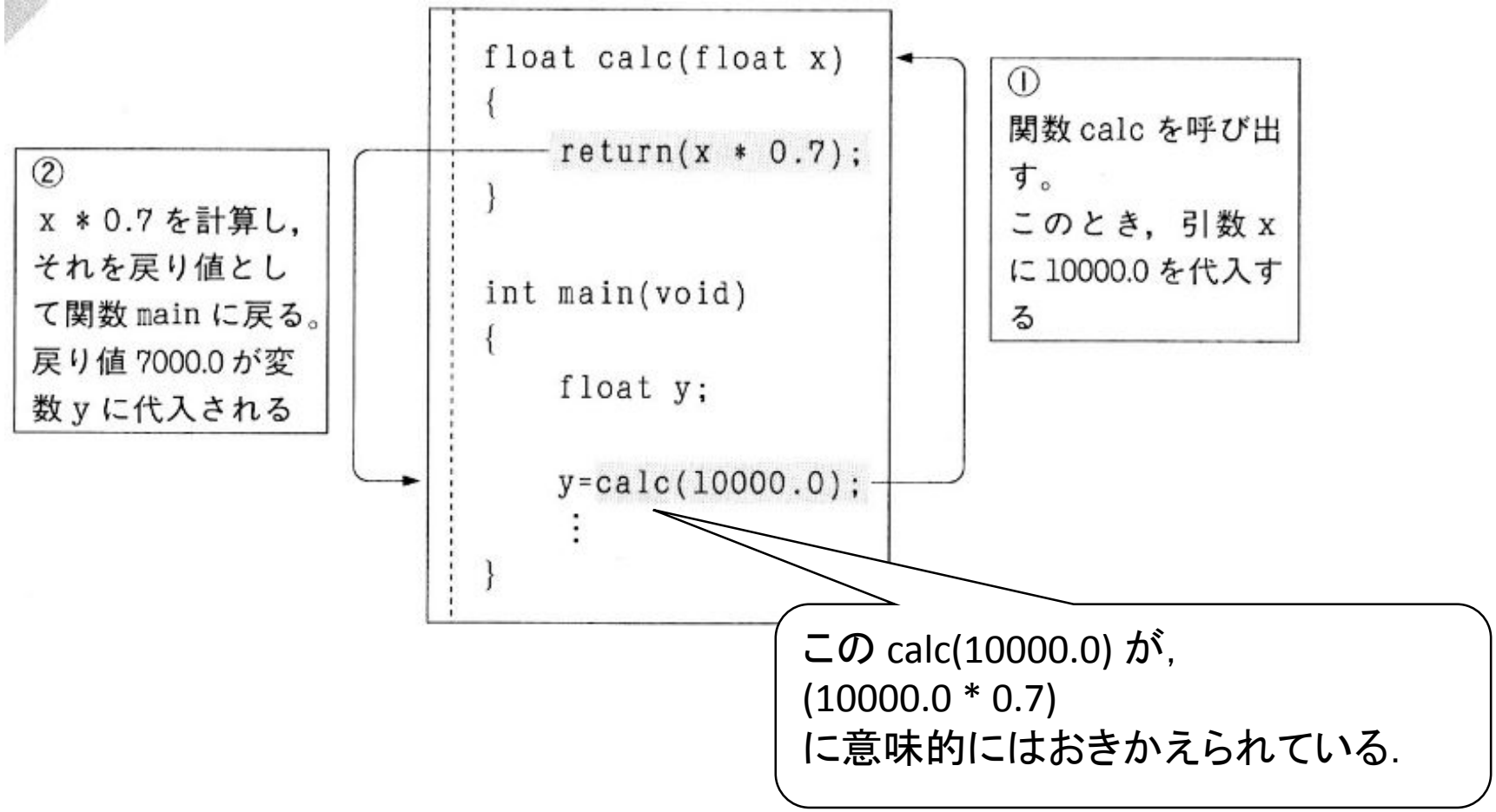


引数と返り値 (戻り値)

- 関数は何かを計算して，その計算結果を返すものが多い.
  - 数学等の $f(x) = x^2+2$ 等は，= の右辺が計算に相当.
- C言語では全てのデータは型(intやdouble等)を決めなければならない.
- よって，関数自体に「返り値の型」が指定されている.

- ある計算をするには，その計算に使うデータ群が必要.
  - 数学等の $f(x) = x^2+2$ では，x が計算に使うデータに相当.
- 関数の引数はこの計算に使うデータ群に相当.
- いくつ使うデータが必要かは計算内容による.
  - 一つも使わない場合もある.

関数の使い方 Fig. 8-3 より



例 関数の仕様

Name
jpeg_read_header -- read start of JPEG datastream

Synopsis
#include <jpeglib.h>

int jpeg_read_header(j_decompress_ptr cinfo, boolean require_image);

Description
The function jpeg_read_header() shall read the JPEG datastream until the first SOS marker is encountered. The function shall initialize all decompression parameters to default values and save all tables and parameters in the decompression object structure.

Return Value
jpeg_read_header() shall return with one of the following return codes:

JPEG_HEADER_OK
  if SOS marker is reached
JPEG_HEADER_TABLES_ONLY
  for an abbreviated input image, if EOI is reached
JPEG_SUSPENDED
  if data source module requests suspension of the decompressor.

Errors
jpeg_read_header() returns error if it encounters end-of-image and require_image is TRUE.

もちょっと身近な関数の仕様

```
名前
       atoi, atol, atoll, atoq - 文字列を整数型に変換する

書式
       #include <stdlib.h>

       int atoi(const char *nptr);
       long atol(const char *nptr);
       long long atoll(const char *nptr);
       long long atoq(const char *nptr);

   glibc 向けの機能検査マクロの要件 (feature_test_macros(7) 参照):

       atoll(): _BSD_SOURCE || _SVID_SOURCE || _XOPEN_SOURCE >= 600 ||
       _ISOC99_SOURCE; または cc -std=c99

説明
       atoi() 関数は、nptr によって指示される文字列のはじめの部分を int 型整数
       に変換する。この振る舞いは、 atoi() 関数がエラーを見つけない点以外は、

           strtol(nptr, (char **) NULL, 10);

       と同じである。

       atol()  関数と atoll() 関数は atoi() と同様の振る舞いをするが、文字列の
       はじめの部分をそれぞれ long や long long に変換する。 atoq() は atoll()
       の古い名前である。

返り値
       変換された値。
```

関数 atoi について

- 教科書にものってます.
- (ASCIIコードで書かれた)数字を表す文字列を,
- 整数(integer)に変換する関数です.
- atoi という変な名前は ASCII to Integer の略語です. (コンピュータ分野この手の略語が多い)

- 例題 atoi.c

Cの関数の仕様の入手法

- ググる
  - 例えば「C言語 fgets 仕様」または「C言語 fgets マニュアル」等.
    - 上記，fgets のところを調べたい関数名にかえてね.
- UNIX系OSの場合，manコマンドで調べる.
  - man fgets
- 図書を見る.
- 他人に聞く

関数利用の注意

- 使う前にちゃんと仕様を読んでおく.
- 特に，限界や制限について，ちゃんと確認しておく.

参考: プロトタイプ宣言

- 教科書[レ]にはある以下の部分は省略していました.
- 以下の部分はmain関数のプロトタイプ宣言と呼ばれる.
- プロトタイプ宣言は「関数を呼び出す際に，引数の数と型，返り値の型」をチェックするためにある.
- よって，main関数を「呼び出す」部分が無い限りプロトタイプ宣言は必要が無い.

```
#include <stdio.h>

int main(void);

int main(void){
    // なんか処理をかく
}
```

参考: stdio.h stdlib.h中のプロトタイプ宣言

- 今まで「呼び出して」きた printf, gets, atoi 等の関数群のプロトタイプ宣言は stdio.h stdlib.h 中にある.
- ファイル頭の #include <stdio.h> は，主に，このプロトタイプ宣言を参照するために書いてある.
- printf in stdio.h

  extern int printf (__const char *__restrict __format, ...);
- gets in stdio.h

  extern char *gets (char *__s) __wur;
- atoi in stdlib.h

  extern int atoi (__const char *__nptr);

以上はCentOS6.5の例. 他のOSでは違うかも.

プロトタイプ宣言での引数の名前

- プロトタイプ宣言では引数の型は省略できないが，名前は省略してよい．
- むしろ，省略したほうがよいという意見もある．

https://www.jpcert.or.jp/sc-rules/c-api08-c.html

```
// l8-2a.c オリジナル
#include <stdio.h>

float calc(float x);

float calc(float x){
  return x*0.7;
}

int main(void){
  float y;
  y = calc(103.0);
return 0;
}
```

```
// l8-2b.c
#include <stdio.h>

float calc(float);

float calc(float x){
  return x*0.7;
}

int main(void){
  float y;
  y = calc(103.0);
return 0;
}
```

関数を作る

- 自分でオリジナル(?)の関数を作成することができる.

1. 名前を決める (既存の関数と違う名前がよい)
2. どのような機能かを決める.
3. その機能に必要な入力に基づき，引数の数と型を決める.
4. 機能に基づき，返り値の型と内容を決める.
   - エラー等が起こった場合も考慮して.
5. 利用制限や限界を決める.
   - それが破られた場合の対処を決める.

- 尚，既に存在する関数名は使わないこと.
  - 存在する関数の機能が上書きされてしまう.

名前のルールと関数内ローカル変数

- 関数名は英語の大文字，小文字，数値，文字 _ を組み合わせて利用できる.
  - 先頭文字は数字はダメ，_ はよい.
- 命名規則は基本，会社や組織毎で決まっている.
  - DQNネーム禁止！
- 原則，動詞(+目的語)に基づく命名がよい.
  - `calc_reduced_price`
  - `calcReducedPrice`

  等

  前者はスネークケース，後者はキャメルケースと呼ばれる命名記法.
- 関数内部だけで利用できる変数が定義できる.
- これは関数定義本体がブロック({}で囲まれた命令群)でできているので当たり前.

教科書 例題8-2

1. 名前 calc
2. 実数値の0.7倍の数値を計算する.
3. 入力されるfloatの実数値.
4. 値の正負に関係なく2.で定めた値をfloatで返す.
5. 内部的にも floatのまま計算するので精度はよくない.

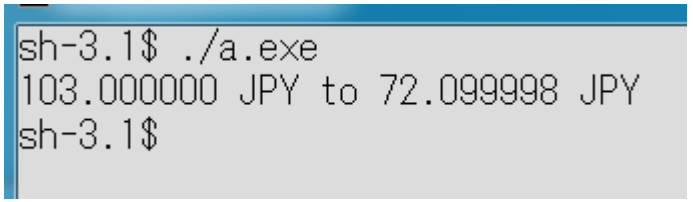
List 8-2

```
// l8-2b.c 引数名をプロトタイプ宣言で省略
#include <stdio.h>

float calc(float);

float calc(float x){
  return x*0.7;
}

int main(void){
float x=103.0, y;
  y = calc(x);
  printf("%f JPY to %f JPY¥n", 103.0, y);
  return 0;
}
```



教科書の例題8-2に似た例

1. 名前 calc2
2. 三割引の金額を計算する関数．金額だから正の整数であり，割引額は切り捨てる．例えば，103円の三割は30.9円だが，切り捨てて，30円引きとする．
3. 入力は金額を示す整数をintで入力する．入力時点では負の数を排除できない．
4. 結果はintで返し，2.に基づく三割引の整数を返す．負の数の場合，割引計算は行わず，その値をそのまま返す．
5. 0以上，intが扱う上限値まで対処できる．負の数の場合，4.に示したとおりに動作する．

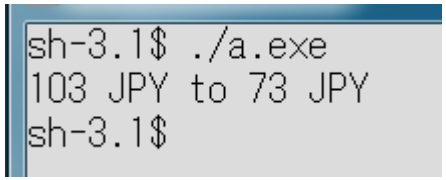
List 8-2 改

```
// l8-2x.c
#include <stdio.h>

int calc2(int);

int calc2(int price){
  if(price < 0 ) return price; // 負の数はそのまま返す
  return price - price*3/10; // 割引額切捨てなので, こうした
  //return price *7/10; // コレだと結果が変わる場合がある
}

int main(void){
  int x=(103), y;
  y = calc2(x);
  printf("%d JPY to %d JPY¥n", x, y);
  return 0;
}
```



プロトタイプ宣言の役割例

- List 8-2 でも List 8-2改でも，calc (もしくはcalc2)のプロトタイプ宣言が行われている.
- これは，main関数中で calc (calc2)が使われているからである.
- Cコンパイラ(ccコマンド)の基本動作は以下の通り.
  1. プログラムを前から順に読み，どのような関数があり，それらの返り値の型と引数の数と型が何かを記録する.
  2. 関数を使われている箇所で，1.で記録した情報とすり合わせを行う.
  3. もし，返り値の型，引数の数や型が一致していなければ，エラーと返す.
- 上記の手順をかんがみると，実は使う関数を前に書いておければ，プロトタイプ宣言は省略できる.
  - しかし，これはかなりズルな方法なので忘れるように.

意図的に誤りを作った例

```
// l8-2er.c
#include <stdio.h>

float calc(float);

float calc(float x){
  return x*0.7;
}

int main(void){
  float y;
  y = calc(103.0, 10.0);
  printf("%f JPY to %f JPY¥n", 103.0, y);
  return 0;
}
```

```
sh-3.1$ cc l8-2.c
l8-2.c: In function 'main':
l8-2.c:12:3: error: too many arguments to function 'calc'
   y = calc(103.0, 10.0);
   ^
l8-2.c:6:7: note: declared here
 float calc(float x){
       ^
```

引数の数があってない！
正解 1個
ココで使う段階で2個かいてある

プロトタイプ無しで関数定義が後

```
#include <stdio.h>

int main(void){ // l8-2noProto.c
  float y;
  y = calc(103.0); // ココで使ってる
  printf("%f JPY to %f JPY¥n", 103.0, y);
  return 0;
}

float calc(float x){ // ココで定義してる
  return x*0.7;
}
```

```
sh-3.1$ cc l8-2.c
l8-2.c:13:7: error: conflicting types for 'calc'
 float calc(float x){
       ^
l8-2.c:8:7: note: previous implicit declaration of 'calc' was here
   y = calc(103.0);
       ^
```

関数定義が後で，使うのが先だと，上記のようにエラーがでる．
定義を先に書くか，もしくは，プロトタイプ宣言をちゃんと書けば問題なし．
定義を先に書いてプロトをさぼるのはお勧めしない．

返り値(戻り値)の無い関数

- 数学の関数で返り値が無いなんて全く意味がない.
- しかし，プログラムの関数では意味があることがある.
  - 関数実行によって，画面に文字を表示する.
  - 関数実行によって，ファイルに何か書き込む.
  - 関数実行によって，通信を行う，等.
- 数学と違い，上記の「画面」や「ファイル」や「通信(回線)」等，引数や返り値とは別に関数実行(計算)の結果に影響を受けるものがある.
- 「画面」等の影響を受ける要素を「環境」，その影響自体を「副作用」と呼ぶ場合がある.
- 戻り値が無いので，return はかかなくてもよい.

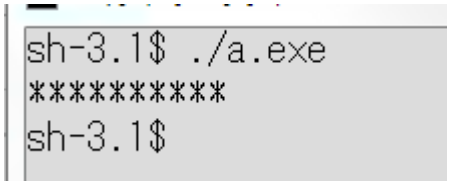
List 8-3 副作用のある関数

```
// l8-3.c
#include <stdio.h>

void print_graph(int);

int main(void){
  print_graph(10);
  return 0;
}

// x個の*を書く関数，戻り値は無い！
void print_graph(int x){ // 意図的にmainより後に書いてみました
  int i=0;
  while(i<x){
    printf("*");
    i++;
  }
  printf("¥n");
}
```



複数の引数指定 and 引数無し

- 複数の引数を持つ関数の場合,
  - 型 引数名

  を , で区切って並べなければならない.
- 引数が無い場合は,
  - 関数名 () ではなく,
  - 関数名(void) とする.
- 関数名() とかくと,「引数の数は不定です. 」の意味になる.

```
// l8-4.c
#include <stdio.h>

int get_power(int, int);

int main(void){
  printf("%d¥n", get_power(8, 2));
  return 0;
}

// xのn乗
int get_power(int x, int n){
  int i=0, y=1;
  while(i<n){
    y *= x;
    i++;
  }
  return y;
}
```

関数内で引数の値は変更できない

- 関数を呼び出す際には，引数の値のコピーを作成し，関数内で計算を行う.
  - call by value と呼ばれる.
- よって，関数内で引数に相当する変数に，いくら代入を行っても，値は変わらない.

```
// cbv.c call by value
#include <stdio.h>

void unUpdateArg(int);

int main(void){
  int val=10;
  printf("%d¥n", val);
  unUpdateArg(val);
  printf("%d¥n", val);
  return 0;
}

// 引数に代入しても値は変わらないデモ
void unUpdateArg(int x){
  x=108*x; // 適当に更新
}
```

参考: 再帰

- 関数定義の中で，その関数自体を呼び出すことができる.
- このような定義を再帰的定義と呼ぶ.
- 自身の定義で，自身を呼び出すと，無限ループに陥るような気がするが，通常は，引数等の値を見て，if文等で，繰り返しを停止するようにプログラミングする.
- 数学における帰納的な定義と同様である.
- 条件分岐等(if文等)によって，必ず，繰り返し自分自身を呼び出さない部分が無いといけない.

サンプル List 8-4 改

```
// l8-4r.c
#include <stdio.h>

int get_power(int, int);

int main(void){
  printf("%d¥n",
    get_power(2,4));
  return 0;
}
```

```
// xのn乗 もとの定義 n<0未対応
int get_power(int x, int n){
  int i=0, y=1;
  while(i<n){
    y *= x;
    i++;
  }
  return y;
}
```

```
// xのn乗，n<0には対応してないよ，もとの定義よりシンプルかも
int get_power(int x, int n){
  // xのn乗 = x * (xの(n-1)乗)
  if(n>0) {return x * get_power(x, n-1);}
  // xの0乗は1
  else {return 1;}
}
```

本日の演習 (演習11)

- 5ページの定義に従い，5個の数値の分散(注意: 標準偏差では無い)を求める関数を作成せよ．関数の宣言は以下．

  double variance(double, double, double, double, double)

- variance()を計算するにあたり，5個の数値の平均を求める以下の関数を定義し，variance 内で利用せよ．

  double average(double, double, double, double, double)

- main関数において5個の実数値を入力し，関数 variance を呼び出し，その計算結果を画面に出力するプログラムとせよ．
- 以下のテストケース(入力例と期待される結果)全ての動作確認せよ．
- ソースプログラム名は variance.c とせよ．

入力	期待される出力
1, 2, 3, 4, 5	2.0
1, 3, 5, 7, 9	8.0
3, 1, 4, 1, 5	2.56

ヒント

- 倍精度実数を scanf で読み出す場合の書き方 (左)
- 複数の数をscanf で読み出す場合の書き方 (右)

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

int main(void){
double a;
  scanf("%lf", &a);
  printf("%lf¥n", a);
  return 0;
}
```

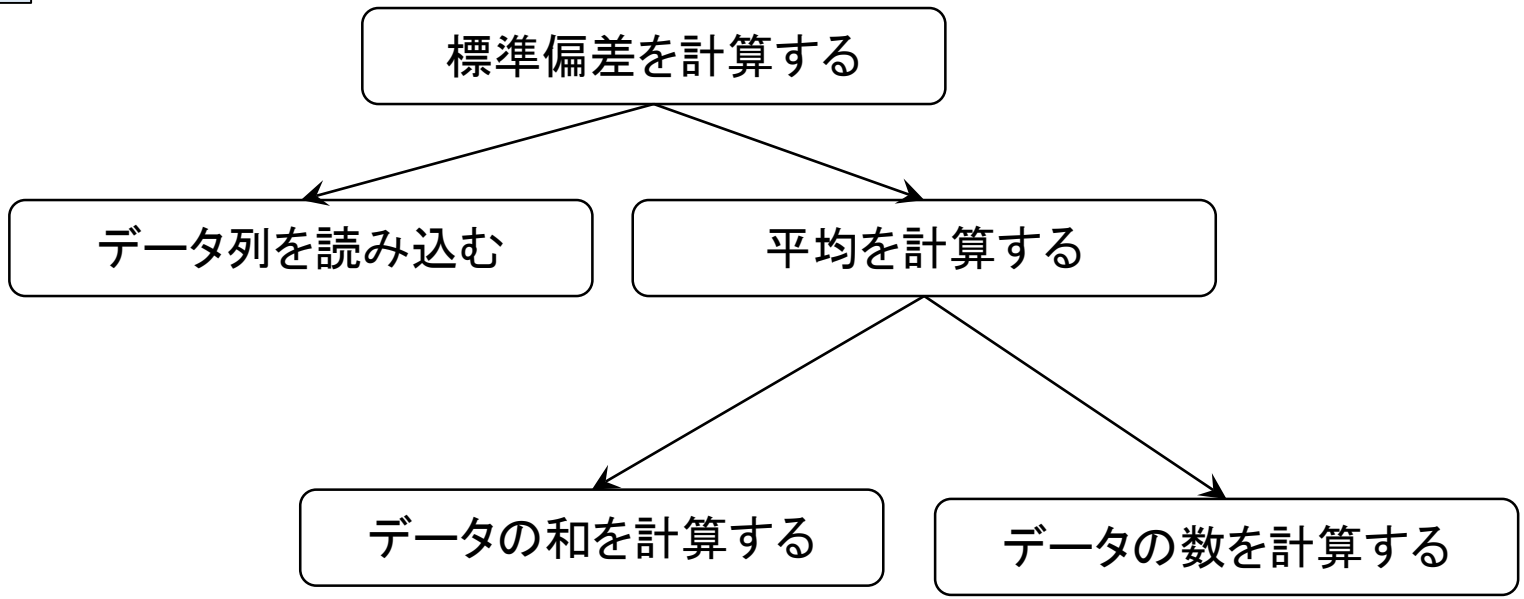
```
// readDoubles.c
#include <stdio.h>

int main(void){
double a, b, c;
  scanf("%lf %lf %lf", &a, &b, &c);
  printf("%lf, %lf, %lf¥n", a, b, c);
  return 0;
}
```

```
bash-3.1$ cc readDoubles.c
bash-3.1$ ./a
1.1 3.2 3.3
1.100000, 3.200000, 3.300000
bash-3.1$
```

5ページ
再掲載

簡単な機能分解の例



データをX1，X2 … Xn、平均値を$\overline{X}$としたとき

分　散　：　$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

標準偏差　：　$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

本日は以上