これまでのまとめ (学年末試験にむけて)

山本昌志*

2004 年 2 月 18 日

1 本日の学習内容

本日の授業では、中間試験以降、学習した内容をまとめる。この間、次のようなことを学習してきた。

- 文字処理
  - コンピューター内部での文字の表現
  - 文字の記憶方法
  - 入出力 (標準入出力とファイル処理)
  - 文字処理のための標準ライブラリー関数
- 関数
  - 関数 (サブルーチン) とは何か
  - 関数 (サブルーチン) の作り方と使い方
  - データの受け渡し方法
    - 戻り値のない場合
    - 戻り値が一つの場合 (値渡しを使う)
    - 戻り値が複数の場合 (参照渡しを使う)
    - 配列を受け渡す場合

学年末試験では、これらの理解度を確かめる。当然、これ以前の内容が分かっていないと、解けない問題もあるので、さかのぼった復習が必要な者も居るであろう。

この間、学習した主な内容は文字処理と関数であるが、後者の法がプログラムを作成する上では重要である。従って、出題の割合は、関数の方が 60〜70%くらいにするつもりである。まずは、関数についてしっかり理解して欲しい。

*独立行政法人　秋田工業高等専門学校　電気情報工学科

2 文字処理

2.1 コンピューター内部での文字の表現

2.1.1 基本事項

まずは、基本的なことである。

- コンピューターで文字を扱うときは、それはすべて、整数に置き換えて取り扱う。文字と整数との対応を決めたものをコード表と言う。
- 1 文字を表すためには、英数字では 1 バイト、日本語では 2 バイト必要である。
- 文字を扱うときには文字型の変数を使う。変数宣言の型は、「`char`」である。これで、宣言された変数は 1 バイトの情報を記憶できる。すなわち、英数字の 1 文字分である。
- 複数の文字が連なったものを文字列と言う。文字列を扱うためには、文字型の配列を使う。
- 文字型の配列に文字列を記憶させた場合、文字列のすぐあとに「\0」が格納される。

2.1.2 文字型変数と代入

つぎに、文字型の変数宣言とそれへの値の代入である。

- 英数字が 1 文字の場合
  - 変数宣言
    ```
    char hoge;              /* 文字型の変数 */
    ```
  - 代入。シングルクォーテーションで囲めば、代入演算子「`=`」が使える。
    ```
    hoge='A';               /* 代入 */
    ```
- 英数字の文字列の場合
  - 変数宣言
    ```
    char hoge[10];              /* 文字型の配列 */
    ```
  - 文字型の配列に文字を格納する方法はいくつかある。代表的な方法は、「`strcpy`」と「`sprintf`」を使う方法である。ただし、前者を使う場合、「`string.h`」をインクルードする必要がある。
    ```
    strcpy(hoge, "Akita");              /* 文字列の格納 */
    sprintf(hoge, "Akita");
    ```
- 日本語の場合は、文字型の配列を使わなくてはならない。必要な配列のサイズは、2× 文字数 +1 である。なぜならば、日本語の 1 文字は 2 バイトで、最後に「\0」を付加するためである。

– 変数宣言

```
char hoge[10];              /* 文字型の配列 */
```

– 文字型の配列に日本語の格納は、英数字と同じで、ダブルクォーテーションで囲む。

```
strcpy(hoge, "秋田");              /* 文字列の格納 */
sprintf(hoge, "秋田");
```

2.2 入出力

文字を取り扱う場合の入出力関数を表 1 に示す。これらの関数を使う場合、以下の注意が必要である。

- 「gets」は改行を除去し、「puts」は改行を付加する。
- 「fgets」は改行を残し、「fputs」は改行を付加しない。
- 「fscanf」では、空白文字 (英数字のスペース) は読み込めない。それは、文字列の区切りを表すからである。

表 1: 入出力に用いられる関数

	標準			ファイル		
	1 文字	1 行	書式付き	1 文字	1 行	書式付き
入力	getchar	gets	scanf	fgetc	fgets	fscanf
出力	putchar	puts	printf	fputc	fputs	fprintf

2.3 標準ライブラリー関数

文字処理のための表 2、文字列処理のための表 3 の標準ライブラリー関数について学習した。

表 2: 1 文字処理関関数。#include <ctype.h>が必要。変数は、int c;。

関数名	動作
isalnum(c)	英数字なら真
isalpha(c)	英文字なら真
iscntrl(c)	制御文字なら真
isdigit(c)	数字なら真
isgraph(c)	印字可能文字なら真
islower(c)	小文字なら真
isprint(c)	空白以外の印字可能文字なら真
ispunct(c)	区切り文字なら真
isspace(c)	空白類文字なら真
isupper(c)	大文字なら真
isxdigit(c)	16 進表示文字なら真
tolower(c)	文字 c を小文字に変換
toupper(c)	文字 c を大文字に変換

表 3: 文字列処理関関数。#include <string.h>が必要。変数は、char s1[256],s2[256]; のように文字型の配列。配列のサイズは、処理に必要なサイズよりも大きいこと (256 とは限らない)。c は文字型の変数、char c; である。

関数名	動作
strlen(s1)	文字列 s1 の長さ、すなわち文字数を整数値返す。
strcpy(s1,s2)	s1 に、文字列 s2 をコピーする。
strcat(s1,s2)	文字列 s1 の後に、文字列 s2 をコピーする。
strcmp(s1,s2)	文字列 s1 と s2 を比較する。 s1 > s2 の場合、戻り値は正 s1 == s2 の場合、戻り値は 0 s1 < s2 の場合、戻り値は負
strncpy(s1,s2,n)	s1 に文字列 s2 の先頭から n 文字をコピーする。
strncat(s1,s2,n)	文字列 s1 の後にと文字列 s2 の先頭から n 文字を連結する。
strncmp(s1,s2,n)	文字列 s1 と文字列 s2 の先頭から n 文字を比較する。比較の結果は、srcmp と同じ。
strchr(s1,c)	文字列 s1 の中の文字 c の位置を整数で返す。文字がないときは、NULL を返す。
strstr(s1,s2)	文字列 S1 の中にある文字列 S1 の位置を整数で返す。もし、文字列がない場合、NULL を返す。

この表を憶える必要は全くない。学年末試験の時には、この表は参考資料として添付する。ただ、この表の関数を使うためには、<ctype.h>や<string.h>をインクルード する必要があることは理解して欲しい。

3　関数

3.1　関数 (サブルーチン) とは何か

3.1.1　数学の関数との比較

関数、正確にいうとプログラマーが作成する自作の関数を学ぶ前は、main 関数１個だけからなるプログラムを作ってきた。単純なプログラムの場合は、それでも良いが、少し複雑になると main 関数以外の自作の関数を使わなくてはならなくなる。実用的なＣ言語のプログラムは複数の関数から構成されている。

関数というものは、まず数学で学習する。入り口と出口に注目すれば、関数というのは一定の個数の値を受け取って，それから決まる値を返す箱[1]のようなものである。その箱は、特定の機能を果たす。数学では、

$$y = f(x_1, x_2, x_3, \cdots, x_n) \tag{1}$$

と書く。$x_1, x_2, x_3, \cdots, x_n$ は独立変数と呼ばれるものである。独立変数の値が決まると、関数 f の定義によって、従属変数がひとつ決まるのである。f という箱に、$x_1, x_2, x_3, \cdots, x_n$ を入れると、ただひとつの値を返すのである。この辺の話は、中学校の数学の時間に関数の定義として、しつこく学習したはずである。これを知らない者は、この学校を卒業できないであろう。理解していないものは、心を入れ替えて勉強しろ。

C 言語の関数は数学の関数と似ている。先ほどの数学の例に倣うと、C 言語では

```
y=f(x1, x2, x3, ... , xn)
```

と記述する。数学では $x_1, \cdots x_n$ を変数と呼ぶのに対して、C 言語を含む手続き型プログラム言語では x1, $\cdots$, xn を引数（ひきすう）と呼ぶ。また関数が返す値を戻値 (もどりち) と呼ぶ。これは、関数 `f` が引数 `x1, x2, x3, ... , xn` の値を受け取って、定義された処理を実施し、戻り値を計算する。その戻り値は、代入演算子 (`=`) により、変数 `y` にコピーされると解釈する。

3.1.2　関数の役割

C 言語のような手続き型プログラミング言語における「関数」の役割は、一連の処理を一まとめにして名前（関数名）で呼び出せるようにすることにある。一般に，プログラムは個々の指令を密接に関連するものごとにまとめて記述することでわかりやすくなる。関数はそのような命令群をプログラム本体から分離して名前をつけたもので，いわばプログラムの部品なようなものである。様々な部品が集まって機械が構成されるように、多くの関数が集まって C 言語のプログラムは出来上がるのである。

全ての関数はどれも優劣が無く、同じレベルにある。それぞれの関数は、互いに呼び出すことにより、それ固有の処理を行う。そして、実際の処理の内容を表す実行文は、どれかの関数に含まれる。従って、処理の内容は関数の中に書かなくてはならない。

ただし、同じレベルにあるが、`main` 関数だけは、少しだけ優位な立場にいる。全ての関数に全く優劣が無ければ、どこからプログラムを実行する順序を決めることができない。そのため、`main` 関数だけ、少し優先順位が高く、最初に実行されることになっている。

---

[1] 入出力のみに注目し、その動作を考える場合、それをブラックボックスと言う

まとめると、関数というのは呼ばれると何らかの仕事をするプログラムのことである。これは一つの大きなプログラムの部品と考えてよい。この関数を用いることにより、プログラムを機能毎に分割できる。そうすることにより、

- 処理の流れが分かりやすいソースプログラムを書くことができる。
- 同じ様な処理を一つにまとめることができる。

というようなメリットがある。

3.2 関数 (サブルーチン) の作り方と使い方

関数をつくり、使うためには、ソースプログラムを図 1 のように記述する。必要な記述は、

- プロトタイプ宣言
- 関数の定義
- 関数のコール (Call:呼び出し)

である。プロトタイプ宣言により、関数の入出力の仕様を示す。そして、関数の定義によりその処理の内容を示す。実際に関数を動作させるために、関数をコールする。

3.2.1 プロトタイプ宣言

関数の定義より前に、必ずプロトタイプ宣言を書く。実際には、コールよりも前に関数の定義を書けば、このプロトタイプ宣言を省くことは可能であるが、それは良くないスタイルである。プロトタイプ宣言はコンパイラに関数の引数の型と個数、それから戻り値を知らせる役割がある。そうすることにより、ソースプログラム中で関数の使い方の間違いをチェックするのである。これは、非常にありがたい機能である。

このプロトタイプ宣言の書き方は、簡単で、実際には関数の定義の先頭部分をコピーして、セミコロンをつければ良い。

プロトタイプ宣言を書くことにより、ソースプログラムを読みやすくなります。現在では、複数のプログラマーによりひとつのプログラムが作成されるため、読みやすいあるいは分かり易いプログラムを書くことは重要である。

プロトタイプ宣言をまとめると

- 書式は、戻り値と関数名、それから引数を順番に書く。
- 関数が正しく使われているか、コンパイラーがチェックするために用意されている。

となる。

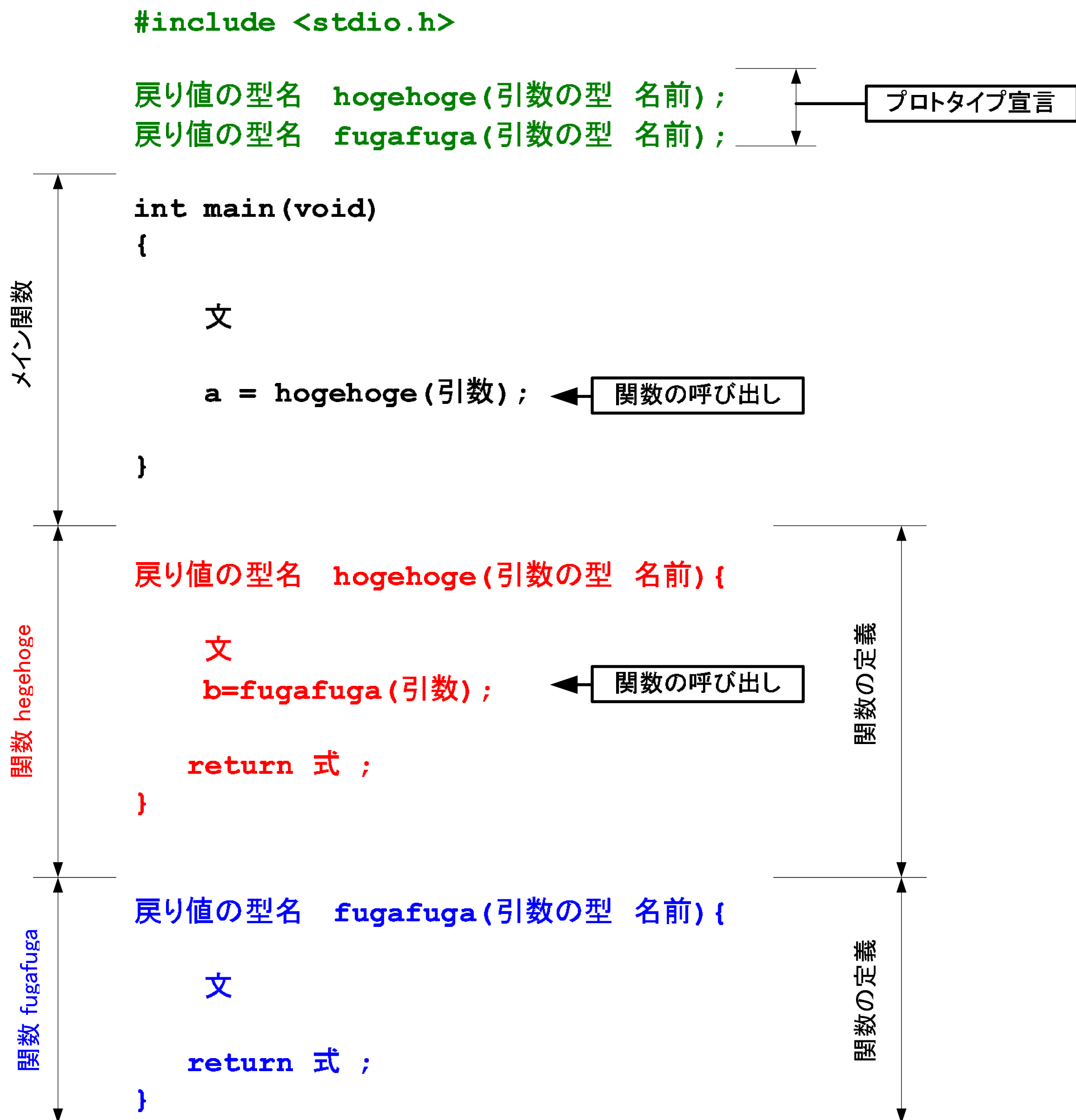

図 1: プログラマーが作成した関数を含んだソースプログラムの書き方

3.2.2 関数の定義

関数は、その動作をプログラマーが定義しないと、コンピューターはどのように処理してよいか分からない。プログラマーによって、ソースプログラム内にその関数の動作を記述する。動作といっても、引数を受け取り、それを処理して、その結果を呼び出し元へ返すと言う内容を記述するだけである。

関数の定義をまとめると、次のようになる。

- メイン関数と同様、処理内容を書けば良い。
- 呼び出し元からのデータは引数で渡される。
- `return` 文で呼び出し元へ、データを送る。その書き方は、次の 2 つが用意されており、どちらでも良い。式は、変数だけでも良い。
  - 括弧無しの方法　`return 式;`
  - 括弧付きの方法　`return(式);`

3.2.3 関数のコール

実際に関数を使う場合、それを使いたい場所で、引数を伴って関数名を書けば良い。関数を使う動作をコールと言う。関数をコールするのは `main` 関数のみならず、他の関数からもコールできる。また、それ自身の関数からもコールできる (再帰呼び出し)。再帰呼び出しについては、2 年生で学習する。

よく使われる処理は、関数にまとめ、必要なときにコールすれば良い。このように、よく使われる処理をまとめると、プログラムの内容が分かり易くなる。関数は、まことに便利な機能である。

- 関数を呼び出すためには、引数を伴って関数名をコールするだけである。
- どこからでも、何回もコールすることができる。

3.3 データの受け渡し方法

関数を使う場合、処理するためのデータの受け渡しがもっとも難しい。これさえ理解すれば、関数は完璧である。

3.3.1 戻り値が無い場合

この場合、プロトタイプ宣言や関数定義の文では、`void`(空っぽの) と宣言する。

```
#include <stdio.h>
void hello(void);            /* プロトタイプ宣言 */

/*================================================*/
/*========      メイン関数           =============*/
/*================================================*/
int main(void){
```

```
  int i;

  for(i=0; i<100; i++){
    hello();         /*  関数呼び出し */
  }

  return 0;
}

/*==================================================*/
/*=======        関数               ==============*/
/*==================================================*/
void hello(void){

  printf("Hello World \n");

}
```

3.3.2 戻り値が一つの場合 (値渡しを使う)

次に示す例は、2 倍角の公式

$$\sin 2\theta = 2\sin\theta\cos\theta \tag{2}$$

を確認するプログラムである。角度を 0〜360 度まで、1 度ずつ変化させてそれぞれを、表示させている。

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

double function1(double x);           /* プロトタイプ宣言 */
double function2(double x);           /* プロトタイプ宣言 */

/*================================================================*/
/*   main function                                                */
/*================================================================*/
int main(void){
  int kakudo;
  double pi, theta, y1, y2;

  pi=3.141592;

  for(kakudo=0; kakudo<=360; kakudo++){
    theta = kakudo*pi/180.0;
    y1 = function1(theta);         /*  関数呼び出し */
    y2 = function2(theta);         /*  関数呼び出し */
    printf("%d\t%lf\t%lf\n", kakudo, y1, y2);
  }
```

```
  return 0;
}


/* <<<<<<<<<< これ以降は、サブルーチン >>>>>>>>>>>>>>>>>>>>>>>>>>>>*/

/*======================================================================*/
/*   sin(2x) の計算                                                     */
/*======================================================================*/
double function1(double x){
  double y;

  y=sin(2.0*x);

  return y;
}


/*======================================================================*/
/*   sin(x)cos(x) の計算                                                 */
/*======================================================================*/
double function2(double x){
  double y;

  y=2.0*sin(x)*cos(x);

  return y;
}
```

3.3.3 戻り値が複数の場合 (参照渡しを使う)

ここで示すようなデータの渡し方を参照渡し (call by reference) と言う。次のような、変数に格納されているデータを交換するプログラムを例にして、参照渡しを考える。

- 2 つの整数をキーボードから読み込み、それぞれを変数 `a` と `b` に格納する。
- `a` と `b` の値を入れ替える。ここで、値を入れ替える専用の関数 `swap()` を使う。
- 入れ替えられた値を表示する。

このような、プログラムでは実引数が 2 個必要で、戻り値も 2 個必要になる。今まで、学習してきた方法だと 2 個のデータを返すことができない。これは、`main` 関数とプログラマーが定義した関数でデータを記憶するメモリーがきっちり分けられて、戻り値が一つしか返せないためである。

メイン関数とプログラマー定義の関数で同じメモリーにアクセスできれば、問題は解決する。いろいろな方法があるが、ここでは後に示すプログラムで、参照渡しという技法を使う。これにより、メモリーの状態が図 2 のようになる。プログラマー定義の関数とメイン関数が同じメモリーを使っているのである。そのため、プログラマー定義の関数で、メイン関数の変数の内容を書き換えることができるのである。

参照渡しを実現するためには、呼び出し元の実引数に&をつけ、呼ばれる関数の仮引数に*をつける。使い方については、プログラムを見て欲しい。

```
#include <stdio.h>
void swap(int *i, int *j);            /* プロトタイプ宣言 */

/*================================================================*/
/*   main function                                                */
/*================================================================*/
int main(void){
  int a, b;
  char temp;

  printf("a = ");
  scanf("%d%c",&a, &temp);

  printf("b = ");
  scanf("%d%c",&b, &temp);

  swap(&a, &b);               /* 関数呼び出し */

  printf("a=%d  b=%d\n", a, b);

  return 0;
}


/*================================================================*/
/*   データの入れ替え                                              */
/*================================================================*/
void swap(int *i, int *j){
  int temp;

  temp = *i;
  *i = *j;
  *j = temp;

}
```

3.3.4　配列を受け渡す場合

これも、本当はちょっと難しいが、参照渡しよりも簡単に思える。ポインターと併せて説明するのが普通であるが、ここではそれを行わない。配列のデータを関数間でどのようにすれば、受け渡せるか、分かって欲しい。本当に細かいことは、2 年生以降とする。

10 人の英語の成績を処理するプログラムを例に示す。これは、10 人分のテストの点数を読み込んで、平均点と各人の平均点からの差を計算するものである。具体的には、次のような動作をするプログラムである。

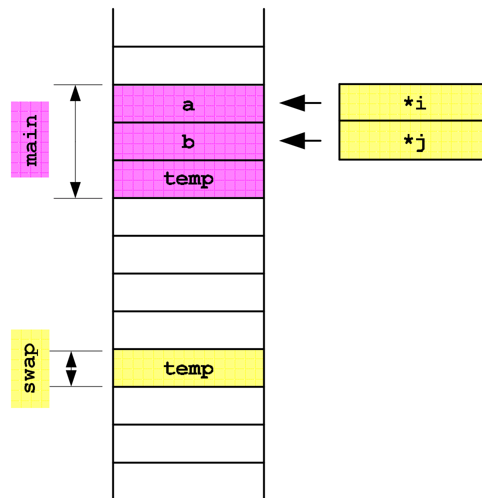

図 2: データの参照渡しのメモリー内容。本当はちょっと違うが、ポインターを使わないでの説明なので勘弁して欲しい。

- 10 人分の英語のテストの点数をキーボードから読み込み、それらを配列 seiseki[0]]〜seiseki[9] に格納する。
- それらを、平均点を計算する。そして、各人の平均点からの差を配列 seiseki[0]]〜seiseki[9] に格納する。これは、専用の関数 diff_ave を用いる。この関数の戻り値は、平均点を示す。
- 最後に、平均点と各人のその差を表示する。

このプログラムの核は、メイン関数とプログラマーが作成した関数 diff_ave() で、同じメモリー上の配列を共有することである。

これは、非常に簡単で、次のようにする。たとえば、呼び出し元で hoge[100] という配列を関数 kansu() に送りたいのならば、

```
kansu(hoge);
```

として、コール (call) すれば良い。一方、呼び出された関数側では、それを配列名 fuga として使いたいならば、

```
戻り値の型 kansu(fuga[]){
  処理内容
}
```

とする。これは、例のプログラムをよく見て欲しい。このようになっているはずである。

2 次元以上の配列の場合も同じようになる。ただし、呼び出された側では、配列のサイズが必要となる。たとえば、呼び出し元で、

```
int hoge[100], fuga[200][300], foo[400][500][600];

kansu(hoge, fuga, foo);
```

とする。そうして、呼び出された関数側では、それを配列名 a,b,c として使いたいならば、

```
戻り値の型 kansu(a[], b[][300], c[][500][600]){
  処理内容
}
```

とする。要するに呼び出された関数側では、配列のサイズの左端を書かなくても良いのである。書いても良いが、一般的には書かない習慣となっている。書かない方が、間違える確率が減るからだろう。

```
#include <stdio.h>

int diff_ave(int n, int data[]);          /* プロトタイプ宣言 */

/*================================================================*/
/*   main function                                                */
/*================================================================*/
int main(void){
  int seiseki[10], average, i;
  char temp;

  for(i=0; i<10; i++){
    printf("%d 番の成績 = ", i);
    scanf("%d%c",&seiseki[i], &temp);
  }

  average=diff_ave(10, seiseki);          /*  関数呼び出し */

  printf("平均点 = %d\n", average);
  printf("平均点との差\n");

  for(i=0; i<10; i++){
    printf("%d 番 = %d\n", i, seiseki[i]);
  }

  return 0;
}

/*================================================================*/
/*   平気値とその差の計算                                          */
/*================================================================*/
int diff_ave(int n, int data[]){
  int i, sum, ave;

  sum=0;                      /* 初期化 */
```

```
  for(i=0; i<n; i++){          /* 点数の合計の計算 */
    sum = sum + data[i];
  }

  ave = sum/n;                 /* 平均値の計算 */

  for(i=0; i<n; i++){          /* 平均値との差の計算 */
    data[i] = data[i] - ave;
  }

  return ave;
}
```

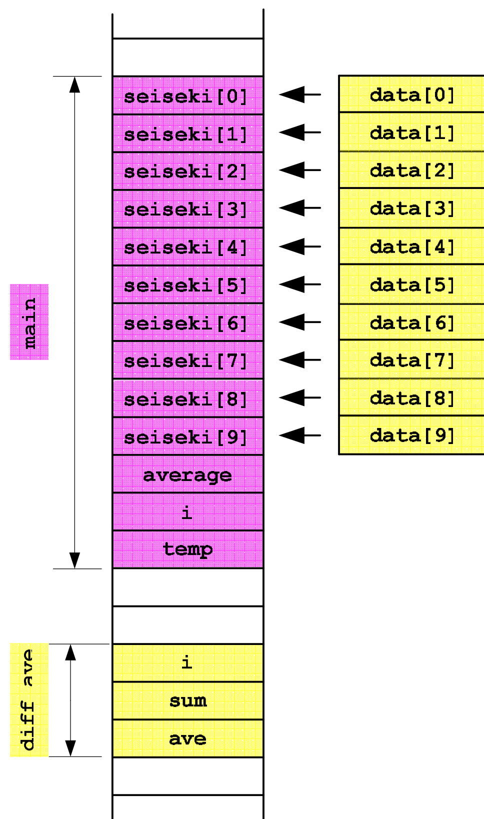

図 3: 配列を受け渡す場合のメモリーの内容。これも、正確ではなく、大体ののイメージである。