関数３

前回までには引数が無い場合と単純型変数の場合、そして返却値が１つだけの場合の関数の使用方法について学んだ。これだけでも多くの種類の関数を作成できるが、この方法だけでは作成できない関数がある。

（１）返却値が複数必要な場合
（２）引数に配列を用いる場合

今回はこれらの使用方法について学習していく。これらの方法を理解するには「アドレス」や「ポインタ」というものを知っておく必要があるが、これらについては後で詳しく学習するので、ここではイメージをつかんだ上で使い方を覚えてもらいたい。

変数の論理的イメージと物理的イメージ

プログラムは変数を利用することでデータを取り扱っている。例えば、

```
int a, b;
a = 10;
b = 20;
```

と宣言された変数 a、b について考えてみる。論理イメージで見れば変数 a に 10、b に 20 が格納されているのだが、物理イメージで見ると以下のように変数 a、b が割り当てられているメモリにデータ 10、20 が格納された状態になっている。

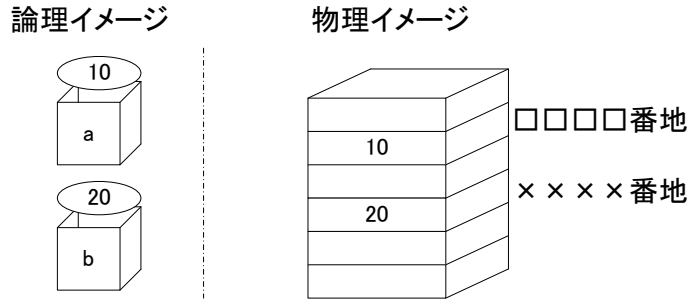


図１　変数のイメージ

C 言語では、この変数が割り当てられているメモリ上のアドレス（番地）の値を利用して、そのデータを取り出したり、格納したりすることもできる。このアドレスを格納するための変数をポインタ変数という。

ポインタ変数の宣言と参照

通常の変数はデータそのものを格納するが、ポインタ変数はアドレスを格納するために利用する。ポインタ変数の宣言方法は以下の書式で行う。

書式
データ型 *ポインタ変数名

使用例
int *pa;

まずは以下のプログラム例を確認しよう。

プログラム例１
#include <stdio.h>

```
int main(void)
{
        int a = 10, b;
        int *pa;

        pa = &a;        //ここからの2行は
        b = *pa;        // b = a;という処理と同じ

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

        return 0;
}
```

出力結果例

```
a = 10
b = 10
```

このプログラムはポインタ変数を介して変数 a の値を変数 b に代入している処理である。pa は int 型のデータを参照するためのポインタ変数である。ポインタ変数を用いてデータを参照するには、ポインタ変数にそのデータが置かれているアドレスを代入する必要がある。そのため、データがメモリの何番地にあるかを知らなくてはならない。よって、

```
pa = &a;
```

と書いて変数 a のアドレスを取得し、ポインタ変数 pa に代入する。変数名に&をつける書き方は scanf 文のところで既に使用している。次に

```
b = *pa;
```

であるが、これはポインタ変数 pa が指すアドレスのデータ（この場合は変数 a のデータ）を変数 b に代入する処理である。よって、この処理は b = a;という処理と同じことをポインタ変数 pa を介して行っている。

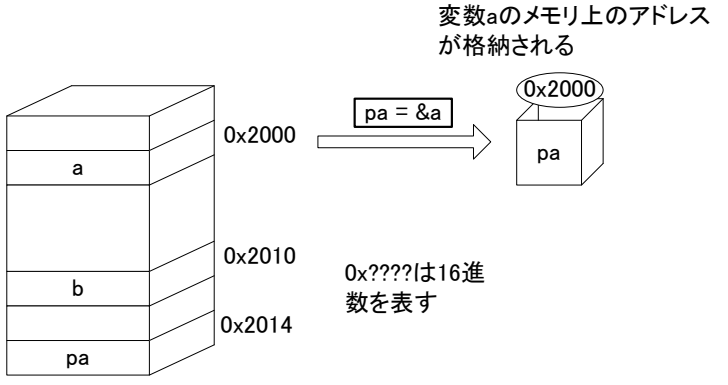


図２　ポインタ変数に格納される値

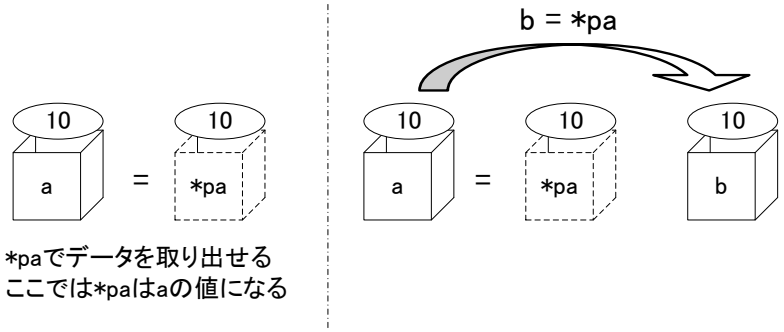


図３　ポインタ変数を介したデータの扱い

配列の場合については例えば、以下のように宣言したとする。

```
int *pa;
int a[20];
```

この場合、a という配列名だけを記述することで配列の先頭アドレスが得られるので、

```
pa = a;
```

というように代入する。ここまでのポイントを以下にまとめておく。

（１） ポインタ変数は「int *pa;」のようにして、「*」をつけて宣言する。 （２） 宣言した後は「pa」と書くとアドレスとなり、「*pa」とするとデータとなる。 （３） 通常の変数は「&a」のようにして、変数名に「&」をつけるとアドレスを取り出せる。 （４） 配列は配列名だけを記述することでアドレスを取り出せる。

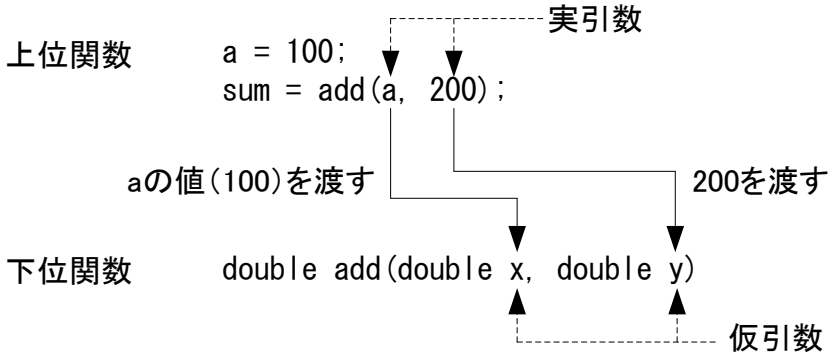
ポインタは文字列、配列のようにメモリ上に連続して置かれているデータを参照する場合や、関数との間でデータをやりとりする場合に威力を発揮する。

関数間のデータ授受の方法

前回で学習したように、関数間では上位の関数（呼ぶ側）から下位の関数（呼ばれる側）へ引数を利用することでデータを渡すことができるが、その方法には、「データの値」を渡す方法と、「データのアドレス」を渡す方法がある。

（１）値による呼び出し（Call by value）

前回の例はこれに相当する。



実引数の値を仮引数に代入するため、実引数と仮引数の変数名は一致しなくても構わない。値が渡されたあと、実引数と仮引数はまったく無関係になり、仮引数の値が変更されても、実引数の値は変わらない。仮に実引数と仮引数の名前が同一でも、両者は連動しない（次の例を参照）。

プログラム例２（関数を用いた値を交換するプログラム：間違った例）

```
#include <stdio.h>

void swap(int a, int b);

int main(void){

        int a, b;
        a = 10; b = 20;

        printf("交換前         :a = %d b = %d¥n", a, b);  // a = 10 b = 20が出力
        swap(a, b);
        printf("交換後<main>  :a = %d b = %d¥n", a, b);  // a = 10 b = 20が出力

        return 0;
}

void swap(int a, int b){

        int t;

        t = a;
        a = b;
        b = t;

        printf("交換後<swap>:a = %d b = %d¥n", a, b);  // a = 20 b = 10が出力
}
```

補足：２つの変数の値を交換するプログラムは以下のように記述する。

```
int main(void)
{
        int a, b, t;
        a = 10; b = 20;
        t = a;
        a = b;
        b = t;
        return 0;
}
```

これは以下のように書きたくなるかもしれないが、少し考えると a、b には同じ値が代入されてしまうことが分かるはず。

```
int main(void)
{
        int a, b;
        a = 10; b = 20;
        a = b;
        b = a;
        return 0;
}
```

（2）参照による呼び出し（Call by reference）

参照による呼び出しでは上位関数で変数のアドレスを渡し、下位関数でそれを受け取る。受け取ったアドレスを介して上位関数側の変数の値を操作する（次の例を参照）。

プログラム例３（**関数を用いた値を交換するプログラム：正しい例**）

```
#include <stdio.h>

void swap(int *a, int *b);

int main(void){

        int a, b;
        a = 10; b = 20;

        printf("交換前         :a = %d b = %d¥n", a, b);  // a = 10 b = 20が出力
        swap(&a, &b);
        printf("交換後<main>  :a = %d b = %d¥n", a, b);  // a = 20 b = 10が出力

        return 0;
}

void swap(int *a, int *b){

        int t;

        t = *a;
        *a = *b;
        *b = t;

        printf("交換後<swap>:a = %d b = %d¥n", *a, *b);  // a = 20 b = 10が出力
}
```

このようにポインタやアドレスの概念を用いることによって、上位関数側と下位関数側でデータを共有し、見かけ上、返却値が複数ある場合について実現できるようになる。

関数間でのデータの授受に配列を用いる方法

関数間で配列を渡す方法について説明する。これは前述の参照による呼び出し（Call by reference）をもちいる（次の例を参照）。ここで、a[]は配列 a の先頭要素のアドレスを示している。アドレスを渡しているため、上位関数（呼ぶ側）で定義した配列の要素にも、下位関数（呼ばれた側）での変更が反映される。

プログラム例４

```
#include <stdio.h>

int add(int a[], int n);

int main(void){

        int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int c;

        c = add(a, 10);
```

```
        printf("合計:%d¥n", c);

        return 0;
}

int add(int a[], int n){

        int i, c;
        c = 0;

        for(i=0; i < n; i++){
                c += a[i];
        }

        return c;
}
```

また、関数の引数に 2 次元配列を用いるには以下のようにすればよい。こちらの場合も前述の参照による呼び出し（Call by reference）をもちいている。渡した引数の 2 次元配列はメモリ上ではあくまでも 1 次元で並んでいる。2 次元配列の先頭要素のアドレスを渡しても、関数側では、何列の配列なのかわからない。そこで列の指定をした 2 次元配列で定義する。行の指定は特には必要ない。

プログラム例５

```
#include <stdio.h>

void disp(int a[][3]);

int main(void){

        int a[2][3] = {{1, 2, 3}, {4, 5, 6}};

        disp(a);

        return 0;
}

void disp(int a[][3]){

        int i, j;

        for(i=0; i < 2; i++){
                for(j=0; j < 3; j++){
                        printf("%d ", a[i][j]);
                }
                printf("¥n");
        }

}
```

演習

プログラム例１～例５を作成し、実行をする。値による呼び出しと参照による呼び出しの出力の違いについて確認すること。配列を引数にした関数の利用方法を確認すること。

課題１の続き

Kadai1-3 （値による呼び出しを用いた関数の復習）

三角形の面積 S をヘロンの公式

$$s = \frac{a+b+c}{2}$$
$$S = \sqrt{s(s-a)(s-b)(s-c)}$$

を使って計算するプログラムを作成しなさい。ここで、a、b、c は辺の長さである。平方根を計算する数学関数は double sqrt(double x)である。使い方の例：y=sqrt(x)； 数学関数の利用のためには#include <stdio.h>の次の行に#include <math.h>と書く必要がある。

プログラムの条件は以下の通りとする。

・キーボードから、辺の長さを読み込む。
・面積の計算は関数を作成して行う。ただし、$s(s-a)(s-b)(s-c) \le 0$ の場合は"三角形になりません"と表示する。
・このプログラムができたならば、再度計算するか否かのメッセージを出す。そして、次の三角形の面積を計算するかプログラムを止めるように変更すること。例えば、「もう一度計算を行いますか？：行う(1) 行わない(0)」と表示して、1 が入力されたならば、再計算、0 ならば終了するようにする。一度は必ず計算するようにしたいので、その場合は do while でループを作成すると良い。

Kadai1-4

次のような配列のデータを循環させるプログラムを作成する。

・ キーボードから整数値を 10 個入力し、配列 a[ ]に格納する。
・ その値を１つ循環させる関数 rotation を作成する。それは 0 番目のデータ→1 番目のデータ、1 番目のデータ→2 番目のデータ、･･･8 番目のデータ→9 番目のデータ、9 番目のデータ→0 番目のデータとさせるものである。
・ main 関数から循環させる関数 rotation を一回呼び出し、その結果を表示。
・ さらに、もう一回呼び出し、その結果を表示。

Kadai1-5

次のような平均値、分散、標準偏差を求めるプログラムを作成すること。

・ 実数でキーボードから数値を 5 個読み込み配列に格納する。
・ メイン関数で平均値、分散、標準偏差を格納する変数を定義し、参照による呼び出しを用いたユーザ関数によって計算する。関数のプロトタイプの例は以下のようになる。

```
void func(int n, double a[], double *ave, double *dis, double *std);
```

・ 引数の最初の n はデータ数であり、次の a[]は入力されたデータである。*ave、*dis、*std はそれぞれ平均値、分散、標準偏差へのポインタである。
・ 平均値＝データの総和÷個数
・ 分散＝（（データ－平均値）の２乗）の総和÷個数
・ 標準偏差＝（分散）のルート

提出締め切りは基本的に 1 週間後の 8：50 まで。