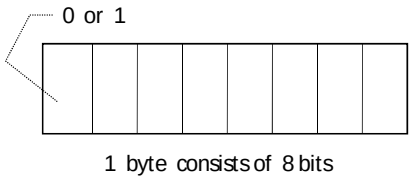
引数の 2 種類の渡し方

[例題 1]　2 つの整数を受け取りその値を入れ替える関数 swap を作ってみよう。

```
 1  /*  正しく動かない swap  */
 2  #include <stdio.h>
 3  void swap (int, int);                  /* プロトタイプ宣言 */
 4
 5  main() {
 6    int i=10, j=20;
 7    swap(i, j);
 8    printf("i=%d, j=%d\n", i,j);
 9  }
10
11  void swap (int m, int n){              /* 正しくない swap */
12    int temp;
13    temp = m;
14    m = n;
15    n = temp;
16  }
```

上のプログラムは期待した仕事をしてくれない。なぜなら、これまで学んだように、引数の受渡しではその値がコピーされるだけで値を受け取った関数側でその値を変えても呼び出した側には影響がないからである。このような引数の受渡し方を「値による呼び出し」(call by value) という。
例題 1 では、その変数の実体を関数間で共有できれば都合が良い。実引数と仮引数の箱を共有するやり方を示す前に、変数の箱がコンピュータの中でどのように実現されているか見てみよう。コンピュータの中は 0 と 1 しかない世界で、その 0 と 1 を取る最小単位をビット (bit) といい、そのビットが 8 つ集まったものをバイトと呼んでいる。



変数の箱はそのバイトをいくつかくっつけて実現されている。ところで下図に示すように、それぞれのバイトには固有の番地 (address) がある。

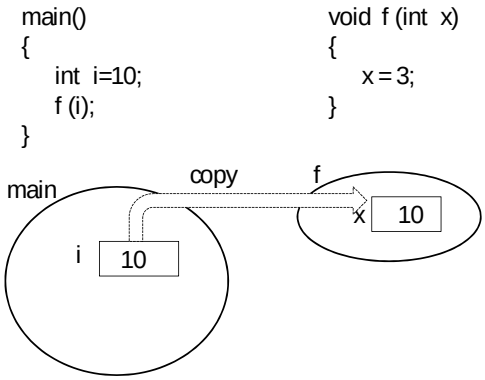
そこで実引数と仮引数の箱を共有するには、この箱の番地を渡してあげればよいことになる。変数の番地を求めるには変数名の前に&を付け、

```
swap(&i, &j);
```

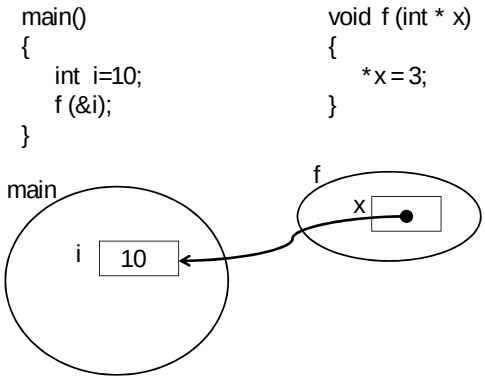
のようにし、受け取る側は

```
void swap(int *i, int *j) { .... }
```

とする。ここで、`int *i` は変数 `i` が整数型の箱を指し示すもの (ポインタ) という意味である。そして、関数内では `*i=1;` のように書いて普通の箱のように使うことが出来る。このように、変数の箱の番地を引数で渡すことを「参照による呼び出し」(call by reference または call by address) という。
次の図は値による呼び出しで引数を渡す状況を示している。この状況において、関数 f 内で `x=3;` を実行した時にどうなるか考えなさい。



下の図は参照による呼び出しで引数を渡す状況を示している。この状況において、関数 f 内で`*x=3;` を実行した時にどうなるか考えなさい。



例題 1 の正しいプログラムは次のようになる。

```
 1  /* 正しく動く swap */
 2  #include <stdio.h>
 3  void swap (int *, int *);
 4
 5  main() {
 6    int i=10, j=20;
 7    swap(&i, &j);                    /* 番地を渡す */
 8    printf("i=%d, j=%d\n", i,j);
 9  }
10
11  void swap (int *m, int *n) {
12    int temp;
13    temp = *m;                       /* *m として普通の箱のように扱える */
14    *m = *n;
15    *n = temp;
16  }
```

ところで、今まで見てきたように関数は 1 つの値しか返すことができない。そこで複数の値を計算してほしいときには、その値のための変数を用意し、参照による呼び出しで渡すとよい。

[練習 1] 整数 i, j を受け取り、i を j で割った商と余りを求めるプログラムを作りなさい。

```
 1  #include <stdio.h>
 2  void warizan (int, int, int *, int *);          /* プロトタイプ宣言 */
 3
 4  main() {
 5    int i, j, q, r;
 6    scanf("%d%d", &i, &j);      /* scanf は変数のアドレスを引数とする */
 7    warizan(i, j, &q, &r);      /* 商と余りのための変数はアドレスで渡し共有する*/
 8    printf("%d / %d = %d ... %d\n", i,j,q,r);
 9  }
10                                              ┌──────┐
11  void warizan (int m, int n, int *quot, │      │) {
12                                              └──────┘
13    *quot = m / n;
14    ┌──────┐
15    │      │= m % n;
16    └──────┘
17  }
```

[練習 2] 整数変数 i のアドレスを受け取り、そこに入っている数を二乗する関数 foo を作りなさい。

[練習 3] 整数変数 i と j のアドレスを受け取り、そのアドレスに入っている数同士を掛けた数を返す関数 bar を作りなさい。

配列のポインタ表記

配列の全要素の和を求める (配列の最後の要素は 0 とする) プログラムを考えてみよう。これまでの知識では、

```
#include <stdio.h>
main () {
  int i, wa=0, a[ ]={1,2,3,4,5,0};

  for(i=0; a[i]!=0; i++)
       wa = wa + a[i];

  printf("array sum = %d\n", wa);
}
```

と書ける。

(注 1) 配列データの初期化をする時は、配列の宣言時に

```
   static int a[ ] = {32, 11, 90, 54, 29, 82, 0};
```

のようにする。
(注 2) これまでの授業で、

```
    if (n != 0)
         { 文 1; 文 2; ... }
    while (i >= 0)
         { 文 1; 文 2; ... }
    for (i=1; i<=10; i++)
         { 文 1; 文 2; 文 3; .. }
```

のように文のかたまりを { } でくくってきたが、文が 1 つしかない場合は { } を省略してもよい。逆に言えば、if (条件), while (条件), for (初期設定; 条件; 後処理) の後に書ける文は 1 文だけで、複数の文を書く時は { } で囲みブロック化しなければならない。

C 言語では、**配列名はその領域の先頭の番地を表している** (これは約束!)。すなわち、a[0] と書く代わりに*a、a[1] の代わりに*(a+1)、a[2] の代わりに*(a+2)、…と書ける。このようなポインタ表記で上のプログラムを書き換えると、

```
#include <stdio.h>
main () {
  int i, wa=0, a[ ]={1,2,3,4,5,0};

  for(i=0; *(a+i) != 0; i++)
       wa = wa + *(a+i);

  printf("array sum = %d\n", wa);
}
```

となる。ここで、a+i のように番地への足し算をしているが、番地そのものの値への足し算と考えずに、先頭要素から i 個先へのアクセスというふうに考える。

これをもっと簡潔に書こうとして、

```
  for(i=0; *a != 0; a++)   /* これはダメ */
       wa = wa + *a;
```

としてはいけない。なぜなら、a の値を変えて「配列名は配列領域の先頭番地」という約束を破ろうとしているからである。このようなことをしたい時は、

```
#include <stdio.h>
main () {
  int wa=0, a[ ]={1,2,3,4,5,0}, *ptr;

  ptr=a;             /* a の先頭番地をコピー */
  for(   ; *ptr != 0; ptr++)
       wa = wa + *ptr;

  printf("array sum = %d\n", wa);
}
```

のようにする。

配列の関数間の受渡し

配列全体を関数の引数にする時は、

```
        main() {
              int  a[10];
              foo(a);                    /* 呼び出し側は配列の名前だけを書く */
        }

        void foo(int *b) {
               .
               .
            tmp = *(b+i);
               .
               .
        }
```

のようにする。これまでの学習では、変数の引数は値がコピーされるだけで、関数内でその値を変えても呼び出した側には影響がなかった。しかし、配列全体を引数で渡す時はその配列自体を共有するので、影響が残る。この違いは非常に重要である。

```
        void foo(int *b) { .......... }
```

という箇所は

```
void foo(int b[ ]) { .......... }     /* 仮引数では要素の数は書かない */
```

としてもよい。さらに、関数内でも

```
tmp = *(b+i);
```

の代りに

```
tmp = b[i];
```

としてよい。

[練習 2]　整数型の配列を引数に取り、配列の各要素の和を返す関数 arraysum を作りなさい。ただし、配列の最後の要素は 0 であるとする。