pointer

他の変数のアドレスを

記憶する変数

変数の2つの意味
記憶場所と値

```
int a, b;

a = 1;

b = a;
```

右辺のaは変数aの値：右辺値

左辺のbは変数bの記憶場所：左辺値

pointer型 *
他の変数のアドレスを記憶する変数

```
int *p;        // p is pointer to int.
               // *：ポインタ宣言子（派生型宣言子の１つ）
int x = 7;     // x is int.   Value of x is 7.
p = &x;        // p is pointer to x.      &：アドレス演算子
               //  これ以降 *p は  x  に等しい。
               //                ↑*  間接参照演算子
               //       ポインタ宣言子の * ではない!
```

ポインタ型変数の宣言の構文は，変数が現れる式の構文を真似たものである。
つまり　int *p　は　*p　が　int であることを思い出させる。
（カーニハン／リッチー　p.114）

pointer型宣言の記述は,
pointerが式で現れる際の構文（ *p)を思い出させるように定義した！！

Pointer型変数の宣言
データ型 *ポインタ変数名;

データ型 *ポインタ変数名；

int *ptr;　// 整数型へのポインタ変数ptr

```
int *ip;          int *xip;

float *fp;        float *xfp;

double *dp;       double *xdp;

char *cp;         char *xcp;
```

変数名の命名法

データ型の頭文字 + p

Pointする変数の頭文字
+ データ型の頭文字
+ p

アドレス演算子　&
間接参照演算子　*

p.141 例題8-2

```
1: int a = 100, b, *ptr;
2: ptr = &a;            // &a : a の記憶場所（アドレス）
                        // 以降 ptr は a を指す。*ptrはaの別名
3: b = *ptr;   // ptrの値（aのアドレス）に記憶されている値
```

派生型宣言子

配列型	関数型	ポインタ型
[ ]	( )	*
int a[ ];	int func( );	int *p; int* p: int * p;

派生型の宣言子は直前または直後の識別子に対してだけ有効

```
int *a, b;          // *は a に作用し，b に作用しない。
int *a, *b;         // a も b も整数型ポインター
```

ポインタ型変数の宣言の構文は，変数が現れる式の構文を真似たものである。
つまり宣言　int *p　は　*p が int　であることを思い出させる。

（カーニハン／リッチー　p.114）

pointer型宣言の記述は，
pointerが式で現れる際の構文（ *p ）を思い出させるように定義した！！

識別子(変数名，関数名）に使える文字

1. アルファベット
2. 数字
3. _

```
int *p;
```

int ：型宣言子

* ：派生型宣言子

p ：変数名

関数の引数

call by value
値による呼び出し
関数内にコピーができる

call by reference
参照による呼び出し
関数内で元の引数を操作できる

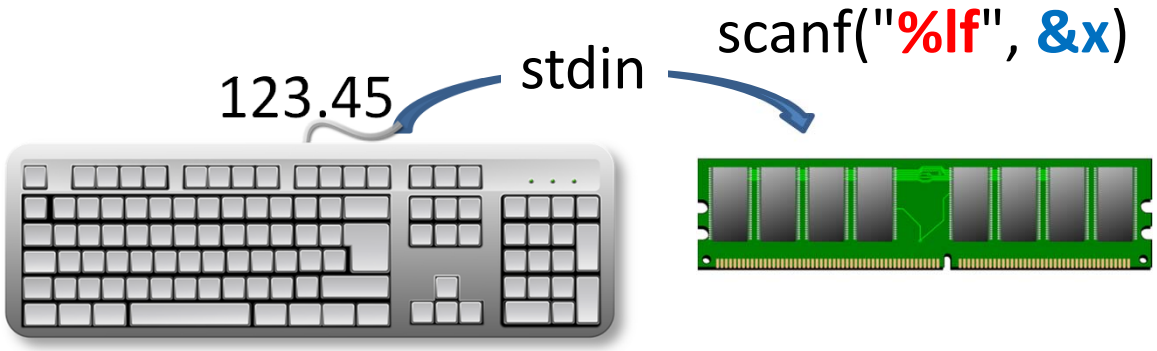
C は　call by value

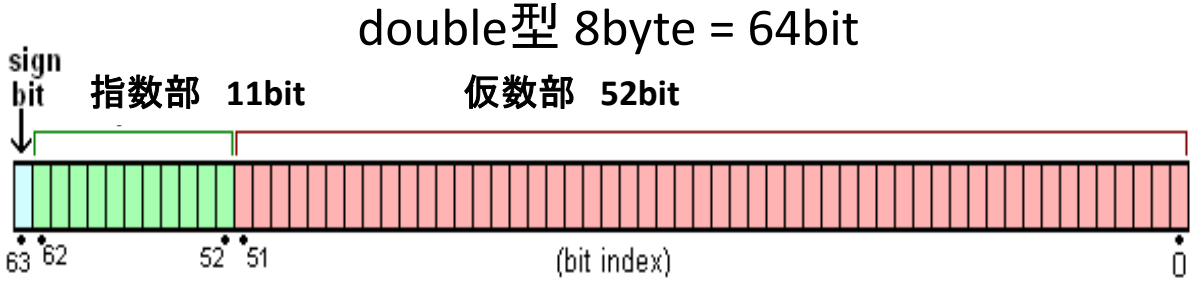
大規模プログラム開発を考えると，変数に局所性がある方が開発しやすい。なぜならば，重要な変数をどこからでも変更できると，予期しない実行結果になるリスクが高まる。また，**pointerを使えば，関数内から外部データを操作できる**。

scanf

```
double x;   scanf("%lf", &x);
```

キーボードからの入力は標準入力ストリーム stdin に文字列として蓄えられ，scanf で読み込み，メモリ内のポインタで指定されたアドレスに，変換指定に従って書き込まれる。





複合宣言子

[ ], ( ) は ＊より優先する。

int *a[3]; // a is array(要素数 3) of pointer to int

int (*a)[3]; // a is pointer to array(要素数 3) of int

int *hoge(); // hoge is function returning pointer to int

int (*hoge)(); // hoge is pointer to function returning int