第 10 章
ポインタ

　プログラミングに限らないことですが、学習を進めていく過程において、対象となる事物を見る目は刻々と変化していきます。
　数値などを格納するための《魔法の箱》であった変数（オブジェクト）も、本章からは、記憶域の一部を占有するオブジェクトとして捉え直すことになります。
　そして、いよいよＣ言語習得上の難関の一つといわれるポインタに突入していきます。

10-1 ポインタ

関数の引数

List 10-1 に示すのは、二つの整数の和と差を求めるプログラムです。

関数 *sum_diff* は、*n1* と *n2* に受け取った値の和と差を求めます。しかし、**main** 関数内で 0 に初期化された *wa* と *sa* は、関数 *sum_diff* が呼び出された後も、その値が 0 のままであり、望んでいる結果は得られません。

List 10-1

```
/*
    二つの整数の和と差を求める（間違い）
*/

#include  <stdio.h>

/*--- n1とn2の和・差をsumとdiffに格納（間違い）  ---*/
void sum_diff(int n1, int n2, int sum, int diff)
{
    sum  = n1 + n2;
    diff = (n1 > n2) ? n1 - n2 : n2 - n1;
}

int main(void)
{
    int  na, nb;
    int  wa = 0, sa = 0;

    puts("二つの整数を入力してください。");
    printf("整数Ａ：");    scanf("%d", &na);
    printf("整数Ｂ：");    scanf("%d", &nb);

    sum_diff(na, nb, wa, sa);

    printf("和は%dです。\n差は%dです。\n", wa, sa);

    return (0);
}
```

実行例

```
二つの整数を入力してください。
整数Ａ：57
整数Ｂ：21
和は0です。
差は0です。
```

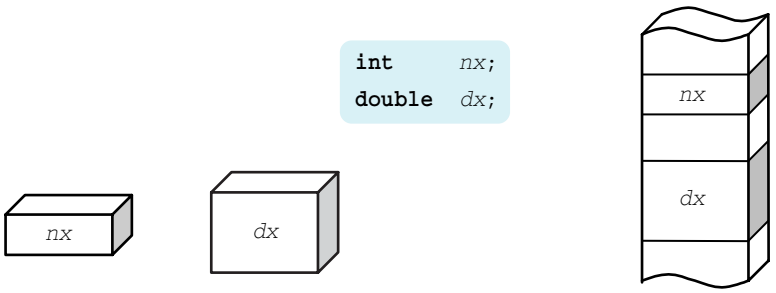
main 関数から関数 *sum_diff* を呼び出すときには、実引数 *na*, *nb*, *wa*, *sa* の**値**が、それぞれ仮引数 *n1*, *n2*, *sum*, *diff* へとコピーされます。このコピーは、一方通行的なものであり、このような引数渡しのメカニズムを**値渡し**と呼ぶのでしたね（*p.120*）。

したがって、関数 *sum_diff* の中で仮引数 *sum* や *diff* の値を変更しても、オリジナルである *wa* や *sa* には、何の変化も与えません。

この問題を解決するためには、C言語の難関の一つである**ポインタ**（*pointer*）を習得しなければなりません。本章では、ポインタの基本について学習していきます。

変数とオブジェクト

「数値などを記憶するための箱」である変数は、**Fig.10-1(a)**のようにバラバラに存在するのではなく、**(b)**に示すように、記憶域（メモリ空間）の一部として存在します。



(a)　箱としての変数　　**(b)　記憶域の一部としての変数（オブジェクト）**

Fig.10-1　オブジェクト

その「変数」には、いろいろな側面すなわち性質があります。たとえば、その性質の一つが大きさです。この図では **int** 型の *nx* と **double** 型の *dx* は、異なる大きさで表現されています。

▶　もちろん処理系によっては、たまたま **sizeof**(**int**)と **sizeof**(**double**)が等しいこともありますが、**第７章**でも説明したように、それを構成するビットの意味が全く異なります。

この性質は、表現できる数値の範囲なども含め、型という概念で捉えられるのでしたね。記憶域上に存在する時間的範囲を表す記憶域期間（**第６章**）、*nx* や *dx* という識別子も重要な性質です。

第２章で簡単に紹介した**オブジェクト**（*object*）は、このように多くの性質や属性をもっているのです。

アドレス

オブジェクトが、記憶域上の《どこに》あるのかを表すのが**アドレス**（*address*）です。アドレスには、“演説”、“住所”、“番地”などの意味がありますが、ここでのアドレスとは**番地**のことであると理解しましょう。ちょうど住所での○○番地と同じようなものです。

■ 重 要 ■

オブジェクトのアドレスとは、それが格納されている記憶域上の番地のことである。

アドレス演算子

各オブジェクトには、アドレスがあるのですから、実際にアドレスを調べて表示してみることにします。**List 10-2**がそのプログラムです。

List 10-2

```
/*
    オブジェクトのアドレスを表示する
*/

#include  <stdio.h>

int main(void)
{
    int     nx;
    double  dx;
    int     vc[3];

    printf("nx   のアドレス：%p\n", &nx);
    printf("dx   のアドレス：%p\n", &dx);
    printf("vc[0]のアドレス：%p\n", &vc[0]);
    printf("vc[1]のアドレス：%p\n", &vc[1]);
    printf("vc[2]のアドレス：%p\n", &vc[2]);

    return (0);
}
```

実行結果例

```
nx   のアドレス：FFCD
dx   のアドレス：FFC2
vc[0]のアドレス：FFB0
vc[1]のアドレス：FFB2
vc[2]のアドレス：FFB4
```

▶ オブジェクトのアドレスが16進数で表示されていますが、処理系や実行する環境によって、基数や桁数などの表示形式や、具体的な数値は異なります。

これまでも使ってきた**単項&演算子**(*unary & operator*)は、一般に**アドレス演算子**(*address operator*)と呼ばれます。**&**演算子をオブジェクトに適用すると、そのオブジェクトのアドレスが得られます(**Table 10-1**)。なお、オブジェクトの大きさが2であって、100番地から101番地にまたがっている場合は、先頭アドレスである100番地となります。

Table 10-1 単項 & 演算子（アドレス演算子）

単項&演算子	**&***a*	*a*のアドレスを得る（*a*へのポインタを生成する）。

▶ 2項の**&**は、第7章で示したビット単位の論理AND演算子ですね。

■ 重 要 ■

アドレス演算子 & は、オブジェクトのアドレスを取り出す演算子である。

そのアドレスを表示するための変換指定は **"%p"** です。

▶ 変換指定 **"%p"** のpは、*pointer* に由来します。

配列*vc*の各要素は、**sizeof(int)**の大きさですから（本書では2と想定)、各要素の先頭アドレスは、**sizeof(int)**だけ離れていることが、実行結果から確認できます。

ポインタ

オブジェクトのアドレスを表示したところで、あまり役に立ちません。もう少し現実的なプログラムを **List 10-3** に示します。

List 10-3

```
/*
    ポインタによって身長を間接的に操作する
*/

#include  <stdio.h>

int main(void)
{
    int  sato   = 178;      /* 佐藤宏史君の身長 */
    int  sanaka = 175;      /* 佐中俊哉君の身長 */
    int  hiraki = 165;      /* 平木Mike君の身長 */
    int  masaki = 179;      /* 真崎宏孝君の身長 */

    int *isako, *hiroko;

    isako  = &sato;         /* isako はsato  を指す（佐藤君が好き） */
    hiroko = &masaki;       /* hirokoはmasakiを指す（真崎君が好き） */

    printf("いさ子さんの好きな人の身長：%d\n", *isako);
    printf("ひろ子さんの好きな人の身長：%d\n", *hiroko);

    isako = &sanaka;        /* isako はsanakaを指す（気が変わった） */

    *hiroko = 180;          /* hirokoの指すオブジェクトに180を代入 */
                            /* ひろ子さんの好きな人の身長を書きかえる */

    putchar('\n');
    printf("佐藤君の身長：%d\n", sato);
    printf("佐中君の身長：%d\n", sanaka);
    printf("平木君の身長：%d\n", hiraki);
    printf("真崎君の身長：%d\n", masaki);
    printf("いさ子さんの好きな人の身長：%d\n", *isako);
    printf("ひろ子さんの好きな人の身長：%d\n", *hiroko);

    return (0);
}
```

▶　プログラムの実行結果は、*p.231* に示します。

変数 *isako* と *hiroko* は、* が与えられて宣言されています。この宣言によって、それらの変数の型は、**int**型のオブジェクトを指すための、“**int**へのポインタ型”となります。

▶　以下のように宣言すると、*hiroko* はポインタでなく、ただの整数となります。

```
int *isako, hiroko;             /* isakoはポインタ、hirokoは整数 */
```

それぞれに * を付けましょう。

```
int *isako, *hiroko;            /* isakoもhirokoもポインタ */
```

まずは、“int 型”と“int へのポインタ型”との違いを明確にしましょう。

int 型のオブジェクト
その値として《整数》を格納する箱。

int へのポインタ型のオブジェクト
その値として《整数を格納するオブジェクトのアドレス》を格納する箱。

ポインタである *isako* や *hiroko* は、その値として普通の《整数》を格納するのではなく、《整数を格納するオブジェクトのアドレス》を格納するのです。

Fig.10-2 には、*sato* の値が 178 であって、そのアドレスが 100番地 である例を示しています。ここで、

isako = &*sato*;

という代入を行うと、*isako* には *sato* のアドレスである 100番地 という値が格納されます。このとき、

isako は *sato* を指す

といいます。

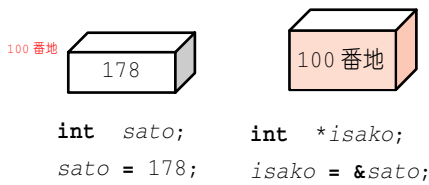


Fig.10-2 int 型と int へのポインタ型

■ 重 要 ■

ポインタ *ptr* の値が、オブジェクト *x* のアドレスであるとき、*ptr* は *x* を指すという。

「指す」という表現ではイメージをつかみにくいので、ここでは

isako は *sato* が好き ♥

と解釈しましょう。続く *hiroko* = &*masaki* の代入によって、

hiroko は *masaki* が好き ♥

となります。ポインタは、オブジェクトを指しますので、**Fig.10-3**のように表せます。もちろん、矢印は好きな人を指しています。

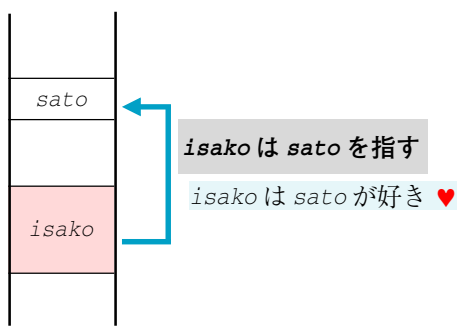


Fig.10-3 ポインタ

▶ *isako* の型は“int へのポインタ型”です。

isako = &*sato*;

という代入からもわかるように、代入する &*sato* の型も、“int へのポインタ型”です。アドレス演算子は、アドレスを得るというよりも、ポインタを生成するのです。式 &*sato* は、*sato* を指すポインタであり、その値として *sato* のアドレスをもつのです。

間接演算子

表示を行う箇所では、**間接演算子**（*indirect operator*）と呼ばれる**単項＊演算子**（*unary * operator*）が登場します。

```
printf("いさ子さんの好きな人の身長：%d\n", *isako);
```

ポインタに間接演算子＊を適用すると、それが指すオブジェクトそのものを表すことになります（**Table10-2**）。

したがって、*isakoは、isakoの指しているオブジェクト（いさ子さんの好きな男の子の身長）の同義語となります。すなわち、

*isako とは sato のことである

であり、*isakoはsatoの**エイリアス**（**別名／あだ名**）となります。

本書では、**Fig.10-4** のような図でポインタを表すことにします。オブジェクトと点線で結ばれた箱に書かれた名前が、それに与えられたエイリアスを示します。

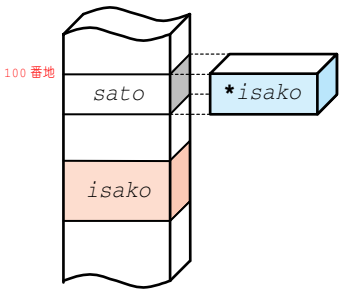


Fig.10-4　ポインタとエイリアス

■ 重 要 ■

ptr が *x* を指すとき、**ptr* は *x* の**エイリアス**（別名）となる。

■ Table 10-2　単項＊演算子（間接演算子）

単項*演算子	*a	aが指すオブジェクト。

▶ 2項の＊は、掛け算を行う算術演算子ですね。

続く代入を考えましょう。*sanaka* へのポインタを *isako* に代入しますので、*isako* は *sanaka* を指すことになります。つまり、

```
isako = &sanaka;
```

isako は sanaka が好き ♥

なのです。*isako*は気が変わったわけですね。このように、他のオブジェクトへのポインタが代入されると、そちらのオブジェクトを指すことになります。

引き続き、少し見慣れない形をした代入です。*hiroko*が*masaki*を指すとき、**hirokoは*masakiのエイリアスですから、**hiroko* に 180 を代入するのは、*masaki* に 180 を代入するのと同じことです。

```
*hiroko = 180;
```

したがって、プログラムの実行結果は、以下のようになります。

```
printf("いさ子さんの好きな人の身長：%d\n", *isako);
printf("ひろ子さんの好きな人の身長：%d\n", *hiroko);
                    /* 中略 */
printf("佐藤君の身長：%d\n", sato);
printf("佐中君の身長：%d\n", sanaka);
printf("平木君の身長：%d\n", hiraki);
printf("真崎君の身長：%d\n", masaki);
printf("いさ子さんの好きな人の身長：%d\n", *isako);
printf("ひろ子さんの好きな人の身長：%d\n", *hiroko);
```

実 行 結 果

```
いさ子さんの好きな人の身長：178
ひろ子さんの好きな人の身長：179

佐藤君の身長：178
佐中君の身長：175
平木君の身長：165
真崎君の身長：180
いさ子さんの好きな人の身長：175
ひろ子さんの好きな人の身長：180
```

10-2 ポインタと関数

関数の引数としてのポインタ

ひろ子さんは理想が高く、恋人の身長が180cmより低ければ、その身長を180cmにしてしまう超能力があるようです。ひろ子さんのこのような能力を**関数**として実現することにしましょう。

もちろん、右に示す関数では駄目です。というのも、本章の最初でも示したように、一時的なコピーである関数の仮引数をいくら変更しても、呼出し側の実引数には反映されないからです。

```
void hiroko(int height)
{
    if (height < 180)
        height = 180;
}
```

値を直接変更できないのならば、ポインタを利用することによって、間接的に変更すればよさそうです。**List 10-4** にプログラムを示します。

List 10-4

```
/*
    関数の引数とポインタ
*/

#include  <stdio.h>

/*--- ひろ子さん（180cm未満の身長を伸ばしてくれる） ---*/
void hiroko(int *height)
{
    if (*height < 180)
        *height = 180;
}

int main(void)
{
    int  sato   = 178;  /* 佐藤宏史君の身長 */
    int  sanaka = 175;  /* 佐中俊哉君の身長 */
    int  hiraki = 165;  /* 平木Mike君の身長 */
    int  masaki = 179;  /* 真崎宏孝君の身長 */

    hiroko(&masaki);

    printf("佐藤君の身長：%d\n", sato);
    printf("佐中君の身長：%d\n", sanaka);
    printf("平木君の身長：%d\n", hiraki);
    printf("真崎君の身長：%d\n", masaki);

    return (0);
}
```

実行例

```
佐藤君の身長：178
佐中君の身長：175
平木君の身長：165
真崎君の身長：180
```

関数 *hiroko* が

```
hiroko(&masaki);
```

と呼び出されたときの様子を示したのが **Fig.10-5** です。

関数*hiroko*では、仮引数*height*を、“**int**へのポインタ型”として宣言しています。関数が呼び出されたときに、*height*には&*masaki*すなわち106番地がコピーされますので、ポインタ *height* は *masaki* を指すことになります。

ポインタに間接演算子*を適用すると、そのポインタの指すオブジェクトそのものを表すのですから、**height* は *masaki* のエイリアスとなりますね。

その **height* の値が 180 より小さければ、180 を代入します。**height*への代入は、*masaki*への代入を意味しますので、関数*hiroko*から**main**関数に戻っても、*masaki* には 180 が格納されていることになります。

```
void hiroko(int *height)
{
    if (*height < 180)
        *height = 180;
}
```

このように、関数に対して、変数の値の変更を頼みたいのであれば、その変数へのポインタを渡します。すなわち、

> ポインタを渡しますから、そのポインタが指すオブジェクトに対して、
> いろいろな処理を行ってください。

と依頼するのです。受け取る側の関数では、そのポインタに間接演算子*を適用することによって、そのポインタの指すオブジェクトを間接的に扱うのです。このことからも、*演算子が間接演算子と呼ばれる理由がよくわかりますね。

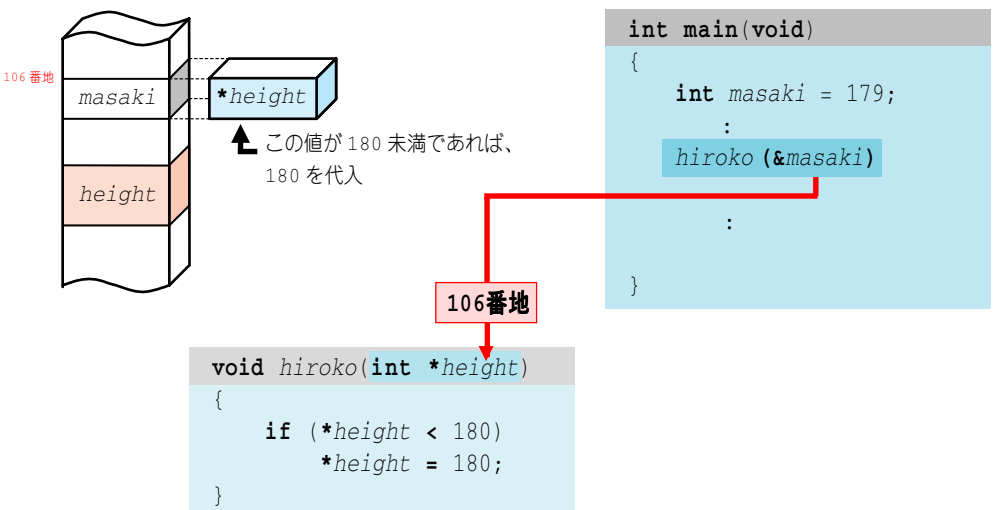


Fig.10-5　関数呼出しにおけるポインタの受渡し

▶　この図は、*masaki* が格納されているアドレスが、106番地であると仮定したものです。今後の図でも、特に断ることなく、適当なアドレスを仮定して示していきます。

二つの値の交換

二つの **int** 型のオブジェクトの値を交換するプログラムを **List 10-5** に示します。

List 10-5

```
/*
    二つの整数値を交換する
*/

#include  <stdio.h>

/*--- nx・nyが指すオブジェクトの値を交換 ---*/
void swap(int *nx, int *ny)
{
    int  temp = *nx;
    *nx = *ny;
    *ny = temp;
}

int main(void)
{
    int  na, nb;

    puts("二つの整数を入力してください。");
    printf("整数Ａ：");    scanf("%d", &na);
    printf("整数Ｂ：");    scanf("%d", &nb);

    swap(&na, &nb);

    puts("これらの値を交換しました。");
    printf("整数Ａは%dです。\n", na);
    printf("整数Ｂは%dです。\n", nb);

    return (0);
}
```

実 行 例

```
二つの整数を入力してください。
整数Ａ：57↵
整数Ｂ：21↵
これらの値を交換しました。
整数Ａは21です。
整数Ｂは57です。
```

関数*swap*が呼び出されたとき、二つの仮引数*nx*および*ny*には、*na*と*nb*のアドレスが格納されます。したがって、**Fig.10-6**に示すように、*****nx*は*na*のエイリアスとなり、*****ny*は*nb*のエイリアスとなります。

*****nx*と*****ny*の値を交換するのですから、結局、**main**関数側の*na*と*nb*の値が交換されることになりますね。

▶ 二つの値を交換する手順に関しては、第 5 章で学習しましたね（*p.95*）。

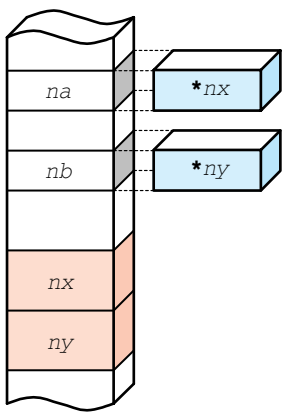


Fig.10-6 関数 swap でのポインタ

参照渡し（C++言語）

C言語から派生した言語であるC++では、引数を値でなく、実体でやり取りすることができます。この機構を、**参照渡し**（*pass by reference*）と呼びます。

参照渡しを用いて書き直したプログラムを **List 10-6** に示します。

List 10-6

```
/*
    二つの整数値を交換する（C++による参照渡し）
*/

#include  <stdio.h>

/*---  nx・nyが指すオブジェクトの値を交換  ---*/
void swap(int& nx, int& ny)
{
    int  temp = nx;
    nx = ny;
    ny = temp;
}

int main(void)
{
    int  na, nb;

    puts("二つの整数を入力してください。");
    printf("整数Ａ：");    scanf("%d", &na);
    printf("整数Ｂ：");    scanf("%d", &nb);

    swap(na, nb);

    puts("これらの値を交換しました。");
    printf("整数Ａは%dです。\n", na);
    printf("整数Ｂは%dです。\n", nb);

    return (0);
}
```

実行例

```
二つの整数を入力してください。
整数Ａ：57↵
整数Ｂ：21↵
これらの値を交換しました。
整数Ａは21です。
整数Ｂは57です。
```

▶　このプログラムをC言語の処理系で翻訳・実行することはできません。

このように実現されると、仮引数*nx*と*ny*は、それぞれ実引数*na*および*nb*のエイリアスとなります。ですから、関数*swap*内では、間接演算子などを使うことなく、呼出し側の*na*および*nb*の値を直接変更する操作が可能となるのです。

▶　参照渡しを用いると、呼出し側の関数と、呼び出される側の関数が、引数を通じて、強く結び付くことになります。このことは、関数の“部品としての独立性”が低くなることを意味します。便利な参照渡しも、使い方を誤ると、プログラムの品質を下げてしまうものなのです（ですから、C言語ではサポートされていません）。

和と差を求める

本章の冒頭で失敗した、二つの整数の和と差を求めるプログラムをどう修正すればよいかは、わかりますね。**List 10-7** に正しいプログラムを示します。

List 10-7

```
/*
    二つの整数の和と差を求める
*/

#include  <stdio.h>

/*--- n1とn2の和・差をsumとdiffに格納 ---*/
void sum_diff(int n1, int n2, int *sum, int *diff)
{
    *sum  = n1 + n2;
    *diff = (n1 > n2) ? n1 - n2 : n2 - n1;
}

int main(void)
{
    int  na, nb;
    int  wa = 0, sa = 0;

    puts("二つの整数を入力してください。");
    printf("整数A：");    scanf("%d", &na);
    printf("整数B：");    scanf("%d", &nb);

    sum_diff(na, nb, &wa, &sa);

    printf("和は%dです。\n差は%dです。\n", wa, sa);

    return (0);
}
```

実行例

```
二つの整数を入力してください。
整数A：57↵
整数B：21↵
和は78です。
差は36です。
```

scanf関数とポインタ

第1章で初めて ***scanf*** 関数を使ったときは、次のように説明していました（*p.12*）。

ただし、***printf***の場合とは異なり、変数名の前に**&**という奇妙な記号を付けなければならないことに注意しましょう。

▶ **&** の具体的な意味などは、**第10章**（*p.236*）で解説します。

scanf 関数は、呼出し側の関数が用意するオブジェクトに対して値を格納しなければなりませんから、変数の《値》をもらっても仕方ありません。ですから、アドレスという《値》をもつポインタを受け取って、そのポインタが指すオブジェクトに対して、標準入力（一般にはキーボード）から読み込んだ値を格納するのですね。

二値を昇順に並べる

List 10-8に示すのは、二つの**int**型のオブジェクトの値を昇順に並べるプログラムです。

List 10-8

```
/*
    二つの整数値を昇順に並べる
*/

#include  <stdio.h>

/*---  nx・nyが指すオブジェクトの値を交換  ---*/
void swap(int *nx, int *ny)
{
    int  temp = *nx;
    *nx = *ny;
    *ny = temp;
}

/*---  *n1≦*n2となるように並べる  ---*/
void sort2(int *n1, int *n2)
{
    if (*n1 > *n2)
        swap(n1, n2);
}

int main(void)
{
    int  na, nb;

    puts("二つの整数を入力してください。");
    printf("整数Ａ：");    scanf("%d", &na);
    printf("整数Ｂ：");    scanf("%d", &nb);

    sort2(&na, &nb);

    puts("これらの値を昇順に並べました。");
    printf("整数Ａは%dです。\n", na);
    printf("整数Ｂは%dです。\n", nb);

    return (0);
}
```

実行例（1）

```
二つの整数を入力してください。
整数Ａ：57↵
整数Ｂ：21↵
これらの値を昇順に並べました。
整数Ａは21です。
整数Ｂは57です。
```

実行例（2）

```
二つの整数を入力してください。
整数Ａ：37↵
整数Ｂ：49↵
これらの値を昇順に並べました。
整数Ａは37です。
整数Ｂは49です。
```

関数*sort2*は、*n1*の指す変数の値が、*n2*の指す変数の値以下になるように並べかえるために、関数*swap*を呼び出しています。*n1*および*n2*は、昇順に並べかえるべき数値が格納された変数へのポインタですから、それらをそのまま関数*swap*に渡すのですね。次のように呼び出してはいけません。

```
swap(&n1, &n2);
```

▶　本書《入門編》の域を超えますので、詳しくは解説しませんが、**&***n1*の型は、“**int**へのポインタへのポインタ”となります。

ポインタの型

右ページ**List 10-9**のプログラムは、二つの**int**型の整数を交換する関数*swap*に**double**型へのポインタを渡すという掟(おきて)やぶり(？)のことをやっています。

プログラムを翻訳するときは、(多くの処理系では)警告メッセージが表示されますし、案(あん)の定(じょう)、実行結果には、変な値が表示されています。

▶ 処理系によっては、たまたまうまく動作するかもしれません。

Fig.10-7を見てください。ポインタ*nx*は、**double**型の*dx*を指していますが、**int**型である***nx*は、**double**型である*dx*のエイリアスとはなり得ないことがわかりますね。

▶ もちろん、**sizeof**(**int**)と**sizeof**(**double**)の大きさが、たまたま同じであっても、ビットの解釈が型によって異なります。

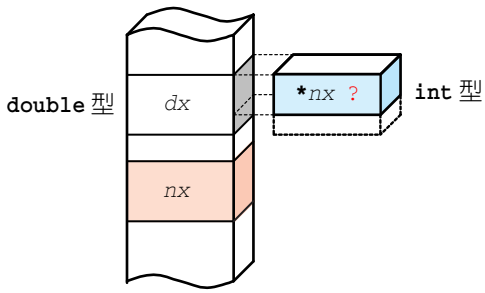


Fig.10-7 ポインタの型と間接演算子

ポインタは、ただ

“○○番地”を指す

のではなくて、

“○○番地を先頭に格納された××型のオブジェクト”を指す

のです。したがって、**int**型のオブジェクトを指すときには**int**へのポインタ型を用い、**double**型のオブジェクトを指すときには、**double**へのポインタ型を用いなければなりません。

10

Column 10-1 アドレスを取得できないオブジェクト

register記憶域クラス指定子(*p.143*)を伴って宣言されたオブジェクトに対して、アドレス演算子**&**を適用することはできません。したがって、次のようなプログラムは、翻訳時にエラーとなります。

```
#include <stdio.h>

int main(void)
{
    register int x;

    printf("%p\n", &x);

    return (0);
}
```

List 10-9

```
/*
    二つの実数値を交換する（間違え）
*/

#include  <stdio.h>

/*---  nx・nyが指すオブジェクトの値を交換  ---*/
void swap(int *nx, int *ny)
{
    int  temp = *nx;
    *nx = *ny;
    *ny = temp;
}

int main(void)
{
    double  dx, dy;

    puts("二つの実数を入力してください。");
    printf("実数Ｘ：");    scanf("%lf", &dx);
    printf("実数Ｙ：");    scanf("%lf", &dy);

    swap(&dx, &dy);

    puts("これらの値を交換しました。");
    printf("実数Ｘは%fです。\n", dx);
    printf("実数Ｙは%fです。\n", dy);

    return (0);
}
```

実行結果例

```
二つの整数を入力してください。
実数Ｘ：53.5↵
実数Ｙ：21.68↵
これらの値を交換しました。
実数Ｘは9980.450456です。
実数Ｙは50.568782です。
```

▶ プログラムの実行結果は、処理系や実行環境によって異なります。

スカラ型

番地を表すポインタは、一種の数量とみなすことができます。**第7章**で解説した算術型と、**本章**で解説したポインタ型をあわせて**スカラ型**（*scalar type*）と呼びます。

● 演習 10-1

西暦 *y* 年 *m* 月 *d* 日の《前の日》あるいは《次の日》の日付を求めてセットする関数

```
void yesterday(int *y, int *m, int *d) { /* … */ }
void tomorrow( int *y, int *m, int *d) { /* … */ }
```

を作成せよ（閏年などもきちんと判別すること）。

● 演習 10-2

三つの **int** 型整数を昇順に並べかえる関数

```
void sort3(int *n1, int *n2, int *n3) { /* … */ }
```

を作成せよ。