10–1 ポインタ

C言語プログラミングで避けて通れないポインタは、『オブジェクトを指す』という特殊な役割が与えられています。本節では、ポインタの基本を学習します。

関数の引数

まず、**List 10-1** を考えます。これは、二つの整数の和と差を求めるプログラムです。

List 10-1 chap10/list1001.c

```
/*
    二つの整数の和と差を求める（間違い）
*/

#include <stdio.h>

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

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で差は0です。
```

ゼロのまま !!

関数 *sum_diff* は、*n1* と *n2* に受け取った値の和と差を *sum* と *diff* に代入します。

main 関数から関数 *sum_diff* を呼び出す際は、実引数 *na*，*nb*，*wa*，*sa* の値が、仮引数 *n1*，*n2*，*sum*，*diff* にコピーされます。値渡しによる引数の受渡しは一方通行（p.140）ですから、関数 *sum_diff* の中で仮引数 *sum* や *diff* の値を変更しても、オリジナルの *wa* と *sa* には、何の変化も与えません。

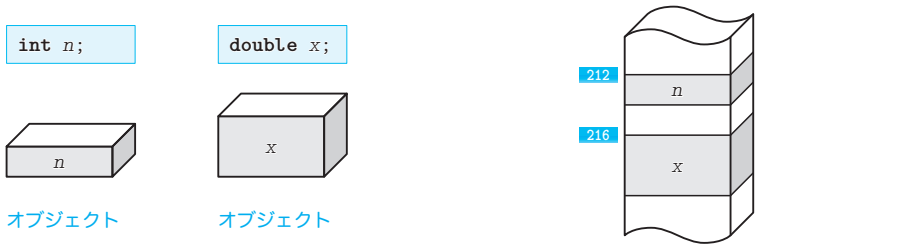
そのため、**main** 関数内で 0 に初期化された *wa* と *sa* は、関数 *sum_diff* が呼び出されて実行された後も、値は 0 のままです。

また、第 6 章で学習したとおり、関数が呼出し元に戻す返却値は 1 個に限られており、2 個以上の値は返せません。そのため、和と差を関数に返却させることもできません。

この問題の解決には、C言語の難関の一つである**ポインタ**（*pointer*）の習得が必要です。本章では、ポインタの基本を学習していきます。

オブジェクトとアドレス

さて、数値などを格納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 バラバラな箱としてのオブジェクト　　b 記憶域の一部としてのオブジェクト

● **Fig.10-1** オブジェクト

その変数には、いろいろな性質があります。たとえば、その一つが**大きさ**です。この図では`int`型の*n*と`double`型の*x*は、異なる大きさで表現されています。それぞれの大きさは、`sizeof`(*n*)と`sizeof`(*x*)で求められるのでしたね。

▶ 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章で学習したように、それを構成するビットの意味が異なります。

表現できる数値の範囲なども含めた**型**も性質の一つです。さらに、記憶域上に存在する生存期間を表す**記憶域期間**（第6章）、*n*や*x*という**識別子**（名前）も重要な性質です。

第2章で簡単に学習した**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ます。

*

広大な空間の記憶域上には、多くのオブジェクトが雑居しています。そのため、個々のオブジェクトの《場所》を何らかの方法で表すことになります。私たちの住まいと同様、場所を表すのは、**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。

図bでは、`int`型オブジェクト*n*のアドレスが212で、`double`型オブジェクト*x*のアドレスが216です。

アドレス演算子

各オブジェクトに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ましょう。**List 10-2** に示すのが、そのプログラムです。

List 10-2 chap10/list1002.c

```
/*
    オブジェクトのアドレスを表示する
*/

#include <stdio.h>

int main(void)
{
    int    n;
    double x;
    int    a[3];

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

    return 0;
}
```

実行結果一例

```
n    のアドレス：212
x    のアドレス：216
a[0]のアドレス：222
a[1]のアドレス：224
a[2]のアドレス：226
```

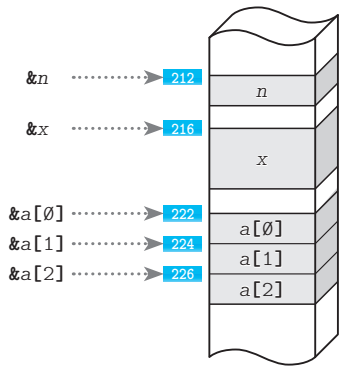


Fig.10-2 アドレスの取得

▶ 実行の結果によって表示されるアドレスの基数や桁数などは、処理系や実行環境によって異なります（通常は、4～8桁程度の16進数です）。

また、本書に示すアドレスの値は、あくまでも一例です。このとおりに表示されるわけではありません。

本プログラムで利用している**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。オブジェクトに **&** 演算子を適用すると、そのオブジェクトのアドレスが得られます（**Table 10-1**）。オブジェクトの大きさが2であって、212番地から213番地にまたがっている場合は、先頭アドレスの212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（aへのポインタを生成する）。

▶ これまでのプログラムでは、***scanf*** 関数に渡す実引数にアドレス演算子を適用していました。なお、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。

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。

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 chap10/list1003.c

```
/*
    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 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("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;
}
```

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
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
```

網かけ部の変数 *isako* と *hiroko* の宣言に着目します。変数名の前に＊が与えられています。この宣言によって、これらの変数の型は、**int** 型のオブジェクトを指すためのポインタとなります。その型名は、以下のように呼ばれます。

- **int 型オブジェクトへのポインタ型**
- **int へのポインタ型**
- **int ＊型**

なお、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ので、要注意です

```
int *isako, hiroko;     /* isakoはint *型ポインタで、hirokoはint型整数 */
```

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**

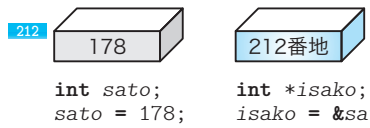
その値として《整数を格納するオブジェクトのアドレス》を格納する箱。

Fig.10-3 の例を考えましょう。int 型の *sato* のアドレスは 212 番地です。そのため、代入“*isako* = &*sato*”を実行すると、*isako* に 212 番地が格納されます（プログラムの大部分を右ページの **Fig.10-6** に再掲しています）。

さて、このとき、*isako* と、それを指すポインタ *sato* の関係を、以下のように表現します。

***isako* は *sato* を指す。**

一般的に表現すると、以下のようになります。



● **Fig.10-3** int 型と int へのポインタ型

重要 ポインタ *p* の値が *x* のアドレスであるとき、『*p* は *x* を“**指す**”』という。

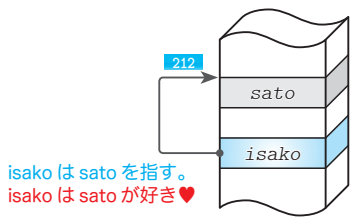
「指す」ではイメージをつかみにくいので、

***isako* は *sato* が好き♥**

という表現を取り入れます。

なお、引き続き“*hiroko* = &*masaki*”の代入が行われるため、以下のようになります。

***hiroko* は *masaki* が好き♥**



● **Fig.10-4** ポインタはオブジェクトを指す

ポインタがオブジェクトを指している様子を表したのが、**Fig.10-4** です。もちろん、矢印の先は、ポインタが好きな男の子（の身長が格納された変数）です。

さて、*isako* の型は“int へのポインタ型”です。

```
isako = &sato;
```

の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、評価して得られる値が *sato* のアドレスです。

重要 Type 型のオブジェクト *x* にアドレス演算子 & を適用した &*x* は、Type * 型のポインタであり、その値は *x* のアドレスである。

間接演算子

表示の箇所では、一般に**間接演算子**（*indirect operator*）と呼ばれる**単項＊演算子**（*unary * operator*）を利用しています。**Table 10-2** に示すように、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します。

すなわち、**isako* は、*isako* が指すオブジェクト（好きな男の子の身長）そのものです。**isako* とは *sato* のことであり、**isako* は *sato* の**エイリアス**（**別名**（べつめい）／**あだ名**）です。

この関係を、本書では、**Fig.10-5** の図で表します。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です。

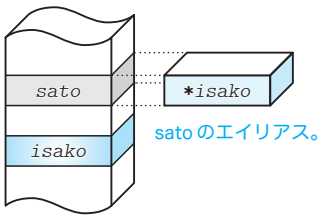


Fig.10-5 間接演算子とエイリアス

重要 ポインタ *p* が *x* を指すとき、**p* は *x* の**エイリアス**（別名）となる。

表示後、“*isako = &sanaka*”の代入によって、*isako* は気が変わります。すなわち、

***isako* は *sanaka* が好き♥**

になります。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、ポインタは、そのオブジェクトを指します（**Fig.10-6** では、*sanaka* は 216 番地としています）。

引き続き行われるのは“**hiroko = 180*”の代入です。*hiroko* が *masaki* を指すとき、**hiroko* は *masaki* のエイリアスですから、**hiroko* への 180 の代入は、*masaki* への 180 の代入と同じことです。

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。

```
isako  = &sato;
hiroko = &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);
isako = & sanaka ;
*hiroko = 180;
putchar('\n');
printf("佐藤君の身長：%d\n", sato);
printf("佐中君の身長：%d\n", sanaka);
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);
```

int 175　　int * 216　　int * 216　　int 175

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
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
```

Fig.10-6 List 10-3 のプログラム主要部分と実行結果