第4章 ポインタ

ポインタとは／ポインタの応用／ポインタと配列／ポインタと2次元配列／
ポインタへのポインタ／ポインタの配列／関数へのポインタ／
配列の動的確保とポインタ／変数の寿命と通用範囲

C言語の特徴の一つにポインタ型変数の取り扱いがあります．「ポインタを制する者はC言語を制す」といわれるくらい，C言語を利用するうえでは重要な事項です．つまりポインタの機能を充分に理解していないと，C言語を使いこなせませんし，効率の良いプログラミングもできません．この章ではポインタに関する事柄を集めて解説します．他の章と一部重複する部分もありますが，ポインタ理解のために例示するものですので，ご容赦ください．

4.1 ポインタとは

ポインタ(pointer)とは指し示すもの，指標のことです．学校で先生が，指し棒を持って黒板を「ここは･･･」といっている姿がありますが，あの指し棒がポインタです．

コンピュータの世界では実際のデータが，「どこどこにある」と指し示しているものです．実際の変数･データはメモリの中にありますから，ポインタ変数の中身はメモリ･アドレスということになります．C言語ではポインタを扱う際に，二つの演算子が重要です．

4.1.1 アドレス演算子

たとえばソース･プログラムの中で，

```
int n;
```

のような変数を宣言したとすると，コンパイラは，

① 名前はnであること
② nの型が整数であること
③ そして，その記憶場所

を管理して，以降のプログラムの機械語への変換を行います．ですから一般に，変数(オブジェクト)そのものはすでに記憶場所の情報，すなわちアドレスをもったものなのです．

変数nのアドレスを返す(表現する)演算子をアドレス演算子(address operator)と呼びます．C言語では &(ampersand)を利用します．具体的に，変数nのアドレスを表現するには，

```
&n
```

と表記します．nそのものは変数の中身･内容を示したのに対し，&nはnが配置されているメモリのアドレスを表しています．&は参照演算子と呼ばれることもあります．

実際にそのことを確認してみます．**リスト4.1.1**では，大局(global)変数u，vと関数内に局所(local)変数a，bを定

リスト4.1.1　アドレスの確認

```
/*  変数の格納アドレスを知る  */
#include <stdio.h>

int u, v;                               /* global 変数             */

void main (void)
{
  int a, b;                             /* local 変数        */

  printf ("global u : %p¥n", &u );      /* アドレスを表示      */
  printf ("global v : %p¥n", &v );
  printf ("local  a : %p¥n", &a );
  printf ("local  b : %p¥n", &b );
  return 0;                             /* 戻り値            */
}
```

```
global u : 0x405040
global v : 0x405050
local  a : 0x22feec
local  b : 0x22fee8
```

図4.1.1　アドレス確認の実行結果

義しました．そしてそれらを，アドレス演算子を使ってポインタとして表示しています．printf文の書式制御の%pは数値をポインタ(アドレス情報)として，16進数で表示するの意味です．

実行結果を**図4.1.1**に示します．ここで留意したいのは，global変数とlocal変数のアドレスが大きく違っていることです．これはglobal変数が，いわゆるデータ領域に確保され，local変数はスタック領域に確保されているためです．変数の確保される領域と，寿命など，いわゆるスコープについては4.9節を参照してください．

この結果から，**図4.1.2**に示すような領域が確保されていることが推測できます．確保される領域の大きさなどの詳細についての説明は割愛します．この&の演算子は，すでに2.7節でscanf()関数利用の際に，入力されたデータを，どこどこに格納するという形で利用しています．

4.1.2　間接参照演算子

C言語では，ポインタ，すなわちアドレスの情報を管理する手法がもう一つあります．アドレスそのものを格納する変数です．便宜上，ポインタ変数と呼ぶことは前述しました．

□ ポインタ変数の宣言

アドレスを格納する目的の変数がポインタ変数です．ポインタ変数も変数，すなわち器ですから，使用する前に宣言をしなければなりません．宣言の方法は，たとえば，

```
int *ptr;
```

です．一般形でいえば，

(対象とする変数の型)　*変数名

です．変数名に*(asterisk)を付加して表現します．ここで宣言した変数ptrは，中身にはアドレスを格納する変数ですから，パソコン環境の場合，型に関係なくサイズは通常4バイトです．これでptrにはアドレス情報を格納する準備，用意ができたことになります．名前の規則などは一般の変数と同様です．

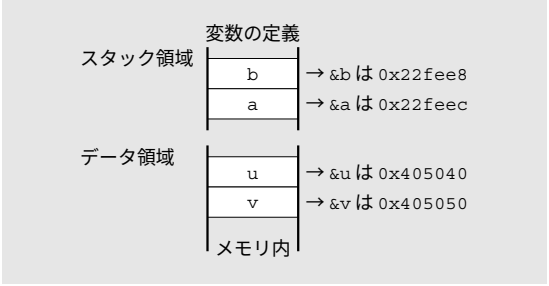

図4.1.2　変数の定義

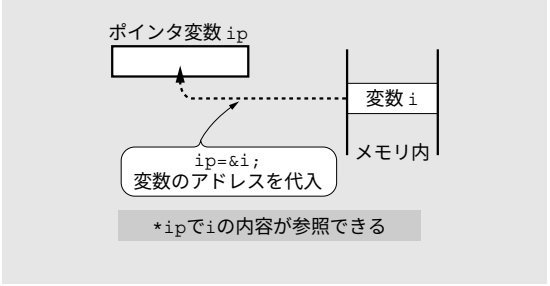

図4.1.3　ポインタによる参照

リスト4.1.2　間接参照の確認

```
/*  ポインタ変数  */
#include <stdio.h>

int main (void)
{
  int   i = 10;                          /* 変数定義・初期設定  */
  int   *ip;                             /* ポインタ定義        */

  ip = &i;                               /* アドレス代入        */
  printf ("初期値    i = %d¥n", i );     /* 初期値表示          */
  *ip = 20;                              /* 間接代入            */
  printf ("代入後    i = %d¥n", i );     /* 変更値表示          */
  printf ("こうしても同じです¥n");
  printf ("代入後  *ip = %d¥n", *ip );   /* 間接参照            */
  return 0;
}
```

```
初期値    i = 10
代入後    i = 20
こうしても同じです
代入後  *ip = 20
```

図4.1.4　間接参照の実行結果

この*については，ちょっとだけ複雑です．この例のようにポインタ変数を定義する目的で使用する場合と，次項で解説する間接的に参照する際に使用する場合があります．本来は同じものなのですが，使用場所・目的で別々に考えたほうがよいかもしれません．

□ ポインタ変数による間接参照

C言語では，文中で前述*をポインタ変数の頭に付加することにより，間接的に参照するという意味になります．つまりポインタ変数はアドレス情報ですが，そのアドレスが指すメモリの内容を参照するという意味です．ポインタ変数が仲介をして，間接的に参照しているわけです．この*を間接参照演算子(indirection reference operator)，あるいはたんに間接演算子といいます．間接指定演算子という呼び名もあります．**図4.1.3**に内容を示します．

間接参照演算子の使用例を**リスト4.1.2**に示します．変数aは定義と同時に10に初期化されています．さらにポインタ変数ipを定義し，

```
ip = &i;
```

と，そこに変数iのアドレスを代入しています．次のprintf()関数でiの初期値を表示すれば当然10です．次に，

```
*ip = 20;
```

と，ipによって間接的にiを指定して20を代入しています．そしてiを表示してみると，20になっています．表示の際に，変数の指定を直接iとしても，間接的に*ipとしても同じです．

このようにポインタ変数の頭に*を付加することによって，ポインタが指している変数そのものの中身を参照できます．実行結果を**図4.1.4**に示します．

□ ポインタは初期化が必要

一般の変数もそうであるように，ポインタ変数も宣言しただけでは，中身はありません．一般の変数は中身への代入がなくても，計算違いが起こる程度ですが，ポインタ変数の場合には，メモリのアドレスを指していることから，初期化を間違いなく行わないと，メモリの破壊，ひいては暴走ということになりかねません．またポインタを更新した場合で，そのポインタを再度利用する場合にも，再初期化が必要です．内容についてはそのつど解説します．

4.1.3　ポインタの基本演算

ポインタ変数に演算を施すことも可能です．ただし元々がアドレスの情報ですから，一般の数値のように四則演算を自在に施すというわけにはいきません．演算の結果がアドレスとしての意味・内容を失わない範囲ならば，加工が可能です．

■ ポインタの1単位

ポインタ変数に関して，まず次のことはしっかり認識しておきます．**リスト4.1.3**に示したプログラムで確認します．このプログラムは，整数の各型の配列を定義し，それぞれの型のポインタ変数を用意して，配列の先頭に初期化しています．そして各型のポインタに，++cpのように増減演算子で更新した場合と，cp+1のようにポインタに＋1

リスト4.1.3　ポインタの1単位

```
/*  ポインタの1単位を調べる  */
#include <stdio.h>

int main ( void )
{
  char  cd[3];                  /* char型     */
  char  *cp = cd;               /* 宣言と初期化*/
  short sd[3];                  /* short型    */
  short *sp = sd;               /* 宣言と初期化*/
  int   id[3];                  /* int型      */
  int   *ip = id;               /* 宣言と初期化*/
  long  ld[3];                  /* long型     */
  long  *lp = ld;               /* 宣言と初期化*/

  printf ("char型では  >> %p",cp);
  printf ("  %p¥n", ++cp);
  printf ("short型では >> %p",sp);
  printf ("  %p¥n", ++sp);
  printf ("int型では   >> %p",ip);
  printf ("  %p¥n", ++ip);
  printf ("long型では  >> %p",lp);
  printf ("  %p¥n", ++lp);
  printf ("¥n");
  printf ("char型では  >> %p %p %d¥n",cp,cp+1,(long)(cp+1)-(long)cp);
  printf ("short型では >> %p %p %d¥n",sp,sp+1,(long)(sp+1)-(long)sp);
  printf ("int型では   >> %p %p %d¥n",ip,ip+1,(long)(ip+1)-(long)ip);
  printf ("long型では  >> %p %p %d¥n",lp,lp+1,(long)(lp+1)-(long)lp);
  return 0;
}
```

した場合，そしてその前との差を表示しているプログラムです．

実行結果を**図4.1.5**に示します．char型ではポインタの値は，0x22fed8と0x22fed9ですから，差は1です．しかしshort型では，0x22feb8と0x22febaですから差は2です．同様にint型，long型では4です．＋1の演算を施したときも同様です．

このようにポインタ変数に対する演算は，その型のサイズを1単位とした演算が行われるのです．配列や構造体・共用体へのポインタとした場合も同様で，やはりそれぞれの型のサイズ(size_t)が1単位になります．

```
char型では  >> 0x22fed8 0x22fed9
short型では >> 0x22feb8 0x22feba
int型では   >> 0x22fe98 0x22fe9c
long型では  >> 0x22fe78 0x22fe7c

char型では  >> 0x22fed9 0x22feda 1
short型では >> 0x22feba 0x22febc 2
int型では   >> 0x22fe9c 0x22fea0 4
long型では  >> 0x22fe7c 0x22fe80 4
```

図4.1.5　ポインタの1単位の実行結果

これは前述のように，変数を定義すると内部的にはそのサイズも記憶・管理され，翻訳の対象になるということの一端です．ですからポインタ変数の修飾だけで，次々と次の要素への指標となり得るわけです．ポインタ変数の便利さの真髄です．

またここで，このプログラムの後半では，その差を得るために，あえて(long)型にキャストして演算している理由についても，注目してください．そのまま演算すれば，ポインタの差だけになってしまいます．下記のように，

```
(lp+1 - lp) * sizeof(long)
```

と，ポインタの差を求めてから，sizeof演算子で各型のサイズを求めて，掛けても同じ値となります．

■ ポインタの演算

ポインタ変数に対して加減算をすることは，その型のサイズを掛けた値を加減算することでした．**リスト4.1.4**で，もう一例見ておきます．このプログラムはint型の配列を用意し，ある値で初期化しています．int型のポインタ変数も用意し，ip0には前述配列のアドレスを，ip1には＋1，ip2には＋2と順に初期設定しています．そしてポインタの数値と，ポインタの指すアドレスの内容を表示しています．

実行結果を**図4.1.6**に示します．ポインタ変数には＋1，＋2とすることで，配列の要素を順に指し示すことができることを示しています．