マイコンを正しく操縦するための作法

基礎から学ぶC言語講座

岡田 好一
Yoshikazu Okada

第3回 ポインタと配列を攻略しよう

C言語におけるポインタは最大の難所のようです．

理解のポイントはいくつかあります．まず当然ですが，出現パターンを知ること．アドレスとポインタと配列の添字の微妙な違いが説明できること．そして独特の記法を正確に読めるようにし，しかし，本人でもすぐにはわからないパズル的記述は書かないことです．

将来のC言語処理系では配列が広範に使えそうですが，現状ではポインタが縦横に活用されています．計算機言語からポインタをまったくなくすのは無理なので，ここはひとつ覚悟を決めて，使いかたを覚えてしまいましょう(**図3-1**)．

ポインタの出現パターン

同じ型のデータが複数あって，次々と処理したい場合にポインタを使用します．ポインタはデータを指し示すので，指し示す先を変えれば同じ式や関数で別のデータが処理できます．つまり，間接指定の一種です．例を示しましょう(**リスト3-1**)．

宣言中の*はC言語のポインタの標識で，変数aがint型へのポインタになります．

式中では，*aの*は間接指定の演算子(単項演算子)です．*aはポインタaの指す先のデータを表します．&は*の逆で，データの主記憶上のアドレス(番地＝位置)を値とする演算子で，上述の場合はint型へのアドレスが値であり，通常はint型へのポインタに代入します．

リスト3-1を実行した結果のRAMの内容を**図3-2**に示します．わざとらしく見えますが，関数の引き数ではこのような使いかた(参照渡しの代用)をします．

ポインタによる複数データ利用の代表例は配列とリストです．

ここでいうリストは，自身のデータ型を指すポイン

図3-1 ポインタの要点
ポインタの攻略にはコツがある

Keyword 1 ポインタ

C言語でなくても，実際上，すべての計算機言語にポインタ(pointer)は存在する．ほかのデータを指すためのデータがポインタである．ポインタ自身の大きさがint型などに一致しているとプログラム作成上は非常に便利．現在のパソコンなど，32/64ビット機の使いやすさの要因の一つである．

R8Cの場合は幸か不幸か，16ビット・アドレスのnearと20ビット・アドレスのfar(32ビット)の区別がある．farポインタがchar型を指すこともあるし，nearポインタがdouble型を指すこともあり，ポインタ自身の大きさとデータの大きさが必ずしも一致しないことがよくわかる．

ややこしいのは言い回しで，ポインタの型とは指す先にあるデータをどのように処理して欲しいか，であって，ポインタ自身の代入のつごうなどは別に考えないといけない．

新規格のC99では，restrictキーワードによって，最適化を促すことができる．本連載で使用のCコンパイラでもrestrictが使えるので，コンパイラの反応を見るのも一興であろう．少なくともライブラリ関数の引き数では役立つと思う．

タが入っているデータ(自己参照的構造体)で，データの連鎖を組み，次々に処理していくしかけです．名人ともなると，枝分かれや合流を駆使した，多種多様なデータ構造をリストで作ります．

配列の基本パターン

配列は同じ処理が見込まれるデータを主記憶上に順に並べて，0から始まる「添字」で指定する複合データ型です．

例えば，n人のクラスで身長を測定して，n個の身長を記録する場合などに使います．学生番号，身長，体重などと項目が複数の場合は，次回で紹介する「構造体」を使うのが自然です．

● intの配列

intの配列，

```
int a[10];
```

と宣言すると，int型(16ビット符号付き整数)のデータが主記憶上に連続で10個確保されます．角カッコ内は「添字」と呼ばれ，参照時に指定します．上述の例だと，a[0]からa[9]まで10個のint型のデータが利用できます(図3-3)．

もちろん，a[3]などと決め打ちでも使いますが，添字に変数が使えるのがミソです．

例えば，同じ演算を繰り返す場合，リスト3-2の

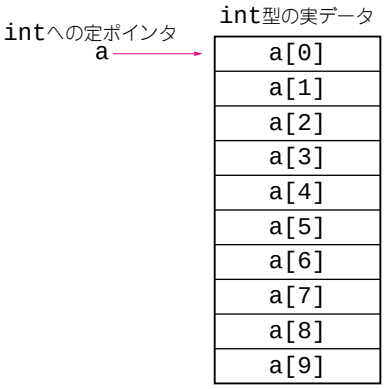

図3-3　int a[10]の宣言でできるメモリ上のデータの配置
aはポインタの定数が書ける位置で使えるが，実体はない

リスト3-1　ポインタを使った間接指定のプログラム例

```
int i, j, k, *a;  // int型を指すポインタ変数aの宣言

void main(void) {
  i = 5; j = 8; k = 99;
  a = &i;         // ポインタaが変数iを指すようにする
  *a = 4 + *a;    // 変数iに4を足す
  a = &j;         // ポインタaが変数jを指すようにする
  *a = 4 + *a;    // 変数jに4を足す
  a = &k;         // ポインタaが変数kを指すようにする
  *a = 4 + *a;    // 変数kに4を足す
   // i == 9, j == 12, k == 103
}
```

Address	Label	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F	ASCII
0400	_SB_	09	00	0C	00	67	00	04	04	FF	FF	FF	FF	FF	FF	FF	FF	....g.....
0410		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
0420		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
0430		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
0440		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
0450		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
0460		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
0470		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
0480		FF	FF	FF	FF	FF	DA	E0	00	FF	FF	FF	FF	FF	FF	FF	FF	.....

図3-2　リスト3-1を実行した結果のRAMの内容
i == 9(0x09)，j == 12(0x0C)，k == 103(0x67)

Keyword 2　参照渡し

私の記憶では，関数の引き数の渡しかたには「値渡し(call by value)」「参照渡し(call by reference)」「名前渡し(call by name)」の3種がある．

値渡しは，実引き数の計算結果で仮引き数を初期化する．つまり，データがコピーされるのが特徴．関数の引き数と言えば，プログラマが真っ先に期待するのはこれ．

参照渡しは，仮引き数の変数名が実引き数の変数名の別名として使えるしかけ．引き数のスコープは値渡しの場合と変わらない．C++で参照渡しが追加されたように，通常の言語には用意されている．参照渡ししかない言語もあるくらいで，むしろ値渡しがなくても困ることはない．ちなみに，C言語の引き数は値渡しのみだが，配列を使うとまるで参照渡しに見える．

名前渡しは，古参の筆者も使ったことがない．ALGOLと呼ばれる言語にあったらしい．話を総合すると，C言語のマクロ(#define)のようなもの．マクロは，もちろん，現役である．