２－１３：ポインタ

プログラミング言語をかじった人は聞いたことがあるかもしれませんが、ポインタは理解が難しいといわれています。制御文のような「方法」「手段」とは色が違い、変数のような「数学」「算数」じみたものではないこともあるでしょう。高度なプログラミングには欠かせない知識ですが、今すぐポインタの意義がわからなくてもあせることはありません。とりあえずどう書くのかだけでも知っておきましょう。

- ポインタはアドレスを扱う物

変数を宣言したとき、変数はコンピュータ内のメモリという場所に記憶されます。そのメモリの記憶する場所の一つ一つにアドレス（番地）がつけられます。

変数が宣言されたとき、適当なアドレスが割り当てられますが、配列の場合連続したアドレスが割り当てられます。

＜例＞

```
int   A,B;
A = 3;
int   C[4] = {10,20,30,40}
```

メモリ内部		
アドレス	名前	データ
100	A	3
130	B	
110	C[0]	10
111	C[1]	20
112	C[2]	30
113	C[3]	40

例えば、この後に　B = A + C[0]；という命令をするとコンピュータは「100 番地のデータと 110 番地のデータを足して 130 番地に記憶する」と解釈します。その結果 B は 13 になります。

- アドレスの表し方

ポインタは割り当てられているアドレスを直接指すことができます。例えば、変数 A のアドレスは&A と表されます。この&A のようなアドレス記憶するにはポインタ変数というものが必要です。

ポインタ変数の名前を PA として宣言するには

```
int *PA;
```

と書きます。A が char 型だった場合は char *PA という風に型を合わせてください。

このポインタ変数 PA に変数 A のアドレスを記憶します。

```
PA = &A;
```

これで「ポインタ変数 PA の指す先は int 型変数 A」となりました。

- アドレスを使った代入

次は変数 B に、先ほどの PA が指す先のデータ（変数）を代入してみます。PA の指す先のデータを表すには*PA と書きます。

```
B = *PA;
```

変数 A のデータは 3 なので、変数 B に３が代入されることになります。

- 配列ポインタ

配列 C[0]において配列名のみ「C」と書けば「&C[0]」として扱われます。配列は連続したアドレスが割り当てられると書きましたが、「C」は配列の先頭アドレスということになります。

ちなみにこの場合、「C」【&C[0]】のアドレスは 110 です。

メモリ内部		
アドレス	名前	データ
100	A	3
130	B	
110	C[0]	10
111	C[1]	20
112	C[2]	30
113	C[3]	40

配列の[]の中の数字は先頭アドレスからいくつ進めるか、という風に考えることができます。例えば C[2]は C(アドレス 110)に 2 を足したアドレス（112）をみるという解釈になります。
そして C[2]は*(C+2)と書いても同じ意味になります。逆も然りです。

これらのことを踏まえると、「ポインタで配列の先頭アドレスさえ渡せば、結果的に配列全てを渡したことになる」ということです。

- コンポーネントを配列のように扱う

ポインタを利用すれば、コンポーネントを配列のように扱うことが可能です。

例えば Shape を 10 個作っていたとします、これらの Shape 全てを非表示にするなら

```
Shape1->Visible = false;
Shape2->Visible = false;
・・・
Shape10->Visible = false;
```

と書かなければなりません。しかしこれをいちいち書くのは面倒ですし、見栄えもよろしくありません。

そこで、ヘッダの変数を宣言する場所に

```
        TShape *Shape[10];   //右の方の名前は自由
```

と宣言し、cpp の最初で

```
//---------------------------------------------------------------------
---------------------------------
__fastcall TForm1::TFrom1(TComponent* Owner):TForm(Owner)
{

        Shape[0] = Shape1;
        Shape[1] = Shape2;
        //・・・・・・中略・・・・・・
        Shape[9] = Shape10;

}
//---------------------------------------------------------------------
---------------------------------
```

これで Shape10 個を配列のようにまとめて使うことができるようになりました。改めて、Shape10 個を非表示にすることを考えて見ましょう。

```
        for(int i=0;i<10;i++){Shape[i]->Visible = false;}
```

これで一気に 10 個の Shape の表示を消せます。

無論 Shape に限らず Label や Buttom でも使えます。

意味がわからなくてもとりあえずプログラミングがスムーズに書けるので是非形だけでも知って使えるようにしてください。

＜例題＞

```
Int A,B;
Int *PA;

A =10;
B =20;
PA = &A;
*PA *= B;
```

上のプログラムを実行したとき、変数Aの値はいくらでしょう。

＜解答＞

```
Int A,B;                    //変数AとB宣言
Int *PA;                    //ポインタ変数PA宣言

A =10;
B =20;
PA = &A;          //変数Aのアドレスを PAに記憶
*PA *= B;         //PAの指す先（つまりA）にBを乗算
//よって 10 * 20 = 200     変数Aは200になる！
```

＜まとめ＞

```
Int A,B;          //変数AとB宣言
Int C[10];        //配列C宣言
C[3] = 398;       //配列Cの先頭＋３のアドレスは398

PA = &A;          //変数Aのアドレスを記憶
PC = C;           //配列Cの先頭アドレスを記憶
B = *(PC+3);      //変数Bに先頭アドレスを代入
//Bは398になる。ポインタ変数PAとPCの宣言は省略した。
```