ポインタとは何ぞや？

Sinryow 特別編集
http://www.sinryow.net/

ActiveBasic でポインタに困っている人のために，ポインタとは何かを説明します。

（注）#N88BASIC や#console 抜きで Print 文を使っている所がありますのでご注意下さい。

1．ポインタとは

現在のコンピュータはほとんど，メモリ（記憶部）にデータを格納する際にメモリに番号（アドレス）をつけてその番号を以ってメモリのどこにデータを書いたり読んだりするかを決定できるようにしています。ポインタとはそのアドレスを指し示すものなのです。
・・・ということはポインタを扱える言語（C など）の本のほぼすべてに書いてあり，それだけだとポインタの意味は分かるわけもなく，かえって混乱してしまいます。そこで実例をいくつか挙げます。

まずは ActiveBasic におけるポインタ処理の基本的な知識として，以下のことを覚えて下さい。

1．アドレスの値は，ポインタ変数という変数に格納する。
2．ポインタ変数は，基本の型に対しては以下の表の通りである。構造体に対するポインタは，構造体の前に「*」を付ける。

基本型	ポインタ型
Byte	BytePtr
Integer, Word	WordPtr
Long, DWord	DWordPtr
Single	SinglePtr
Double	DoublePtr

※（'05.10.1 追記）どのバージョンからか忘れましたが，構造体以外の通常の型についても「*」を付ければポインタ型として扱えるようになっています。（例：「*Byte」は「BytePtr」に等しい）。

3．変数からそのデータを格納しているアドレスを得るには，VarPtr 関数で取得する。

4．ポインタ型が指すアドレスのデータを書き換える関数。

SetByte(BytePtr, Byte)	アドレス BytePtr の位置の Byte 値を Byte にする
SetWord(WordPtr, Word)	アドレス WordPtr の位置の Word 値を Word にする
SetDWord(DWordPtr, DWord)	アドレス DWordPtr の位置の DWord 値を DWord にする
SetSingle(SinglePtr, Single)	アドレス SinglePtr の位置の Single 値を Single にする
SetDouble(DoublePtr, Double)	アドレス DoublePtr の位置の Double 値を Double にする

5．ポインタ型が指すアドレスのデータを取得する関数。

GetByte(BytePtr)	アドレス BytePtr の位置の Byte 値
GetWord(WordPtr)	アドレス WordPtr の位置の Word 値
GetDWord(DWordPtr)	アドレス DWordPtr の位置の DWord 値
GetSingle(SinglePtr)	アドレス SinglePtr の位置の Single 値
GetDouble(DoublePtr)	アドレス DoublePtr の位置の Double 値

例

```
Dim L As Single, pL As SinglePtr
L=10
pL=VarPtr(L) ' pL にはLのアドレスが入る
SetSingle(pL, 20) ' pL のアドレスが指している内容を書きかえると...
Print L ' 元の変数Lの値も変わっている（ここでは「20」が表示される）
```

上の例は簡単なものなので，現実にはこんな使い方だけならポインタは必要ありません。以下で実際にポインタが必要となるケースを説明します。

2. ポインタの使用例1･･･変数の参照渡し

ポインタを用いる1つの用途に，変数を関数に引き渡す際の問題があります。例えば

```
Dim L As Single
L=10
Add20(L)
Print L ' "10"と表示

Sub Add20(Value As Single)
    Value=Value+20
End Sub
```

のようにしても，Add20 関数の呼び出し側で引き渡された変数「L」の値を Add20 関数で書き換えることは出来ません。
これは，Add20 関数は変数 L を受け取ったとは認識せず，ただ単に値としての「10」を受け取ったと認識するからです。
この問題は，実はポインタを用いると解決できます。

```
Dim L As Single
L=10
Add20(VarPtr(L))
Print L ' "30"と表示

Sub Add20(pValue As SinglePtr)
    SetSingle(pValue, GetSingle(Value)+20)
End Sub
```

Add20 関数の引数は Single の代わりに SinglePtr（Single 型へのポインタ）としてあります。先程述べたように，VarPtr(L)は変数 L のポインタ，つまり変数 L がメモリ上のどこに格納されているかを示すのです。その情報があれば，関数側でも変数 L の値を書き換えたりすることが可能なのです。

実際のところ，ActiveBasic には「変数そのものを引き渡すように見せる」方法もあります。

```
Dim L As Single
L=10
Add20(L)
Print L ' "30"と表示

Sub Add20(ByRef Value As Single)
    Value=Value+20
End Sub
```

最初のプログラムと比べて，Add20 関数の引数に ByRef を付けると，引き渡された先の関数でも元の変数の値を変更することができます。ActiveBasic にはそういう「技」もあるのですが，ポインタの説明なので後に廻しました。というよりそもそも ByRef を使っても実際に内部ではポインタを引き渡しているのであり，本来的には ByRef は2番目のサンプルのような面倒な表記を避けるための簡略表記に過ぎないのです。
※C のように ByRef のような指定が出来ない言語では，2番目のサンプルのようなポインタを用いた値の引き渡ししかできません。（ただ，C ではポインタ関係の処理の記述が ActiveBasic に比べてよっぽど簡単に書けるのですが。）

3. ポインタの使用例2･･･配列とポインタ

関数の引数として配列そのものを引き渡したい時は，どうすればよいのでしょうか。そのためには，配列のメモリ上での意味を知る必要があります。
配列変数を宣言すると，それらはメモリ上の連続した領域に配置されます。
例として，「Dim v[3] As Integer」とInteger型4つの配列変数を宣言し，それが仮に先頭のアドレスを150として使うことになったとします。その場合，メモリ上での割り当ては以下のようになります。

アドレス	150	151	152	153	154	155	156	157
対応する変数	v[0]		v[1]		v[2]		v[3]	

Integer型は2バイトですので，150を先頭に合計8バイトの領域が順にv[0]～v[3]に割り当てられていきます。

さて，以下のプログラムを見て下さい。

```
Dim v[3] As Integer
Dim i As Integer
For i=0 To 10
    Print v[i]
Next
```

これは，v[0]～v[3]までしか定義されていないのにv[10]までを取り出すように作ってあります。しかし，これを実行してもエラーには(多分)なりません(なる場合もあります)。
これは，v[3]より先に続くメモリにも配列が続いているとみなして値を取り出し続けているのです。例えば，v[4]は上の例で言えばアドレス158～159にある値を返します(値は何になるか分かりません)。

それでは本題の「関数の引数として配列そのものを引き渡す」ことに入ります。
結論から言うと，配列を引き渡すには，その配列のデータ型のポインタを引数とすればよいのです。
※Integer型へのポインタはWordPtrであることに注意

```
Dim v[3] As Integer
v[0]=20
v[1]=40
v[2]=35
v[3]=30
Print Sum(v, 3) ' v[0]～v[3]の合計を表示

Function Sum(Data As WordPtr, Count As Long) As Integer
    Dim i As Integer
    Sum=0
    For i=0 To Count
        Sum=Sum+Data[i]
    Next
End Function
```

「Print Sum(v, 3)」の部分では関数に配列名を渡していますが，配列名そのもののみを書くとその配列の先頭アドレスを表すポインタ型変数とみなされます。上の例のように配列変数の領域が確保されているとすれば，Sum関数の引数Dataには「150」が入ることになります。

さらに，ポインタ型変数Dataに直接括弧をつけて配列のように扱えることにも着目して下さい。
ポインタ型変数に括弧をつけて値を参照する場合，以下のことが成り立ちます。これは上での配列のメモリ割り当ての意味に一致します。

pをポインタ型変数，TYPEをそれの型名，nを整数値として p[n] = GetTYPE(p + n * SizeOf(TYPE))

※ポインタ型変数の計算の詳細は6で扱います。
※SizeOf(TYPE)とは，TYPEが何バイトを要する型であるかを返します。

4. 配列による文字列の表現(1)

通常 String 型で表している文字列を，文字(Byte 型)の配列とみなして扱う方法があります。(※C などは，char 型(AB の Byte 型に相当する)の配列でしか文字列を扱えません。)

基本的な方法は簡単で，Byte 型の配列に順に文字の文字コードを入れていくのです。ここで注意すべきなのは，文字列の末尾の後ろは 0(″0″という文字ではなく，0 という値)を入れておくことです(場合によっては入れないこともあるのですが，普通使う分には入れておきます)。従って，最低でも(文字数＋1)個の配列要素が必要になります。
例えば，「Sinryow」という文字列を Byte 型配列で表すには

```
Dim str[7] As Byte
str[0]=83  ' ″S″の文字コードは 83
str[1]=105 ' ″i″の文字コードは 105
str[2]=110 ' ″n″の文字コードは 110
str[3]=114 ' ″r″の文字コードは 114
str[4]=121 ' ″y″の文字コードは 121
str[5]=111 ' ″o″の文字コードは 111
str[6]=119 ' ″w″の文字コードは 119
str[7]=0   ' 終端なので 0 を入れる
```

となります。

Byte 型配列には，直接文字列を代入することは出来ません。この場合，lstrcpy という関数があるのでそれを使います。これは文字列ポインタ2つを引数に取り，前者に後者をコピーすることを表します。第2引数には文字列定数や String 型変数も許されます。
※配列名そのもののみを書くと，その配列の先頭アドレスを表すポインタ型変数とみなされることに注意。

```
Dim str[7] As Byte
lstrcpy(str, ″Sinryow″)
str=″Sinryow″ ' これはエラー
```

ただ，下のように BytePtr 型については特別に文字列を直接代入することが認められています。

```
Dim p_char As BytePtr
p_char=″Sinryow″ ' エラーにならない
```

これは，メモリ上のどこかに″Sinryow″という文字列(実際には，文字 7 文字＋終端の 0 の計 8 バイトの領域)があって，その先頭アドレスを p_char に代入することを表します。

Byte 型配列による文字列の比較には，lstrcmp 関数を使います。
同じであれば 0 が返ります。前者が後者より小さい時は負数，前者が後者より大きい時は正数が返ります。
lstrcmp の代わりに lstrcmpi とすると，大文字小文字を区別せずに文字列を比較します。

```
Dim str1[7] As Byte
Dim str2[8] As Byte
lstrcpy(str, ″Sinryow″)
lstrcpy(str, ″Software″)
If lstrcmp(str1, str2)=0 Then
    Print ″同じです″
Else
    Print ″違います″
End If
```

Byte 型配列による文字列の連結には，lstrcat 関数を使います。lstrcpy と同様，第2引数には文字列定数や String 型変数も許されます。
また，Byte 型配列による文字列を String 型の文字列にする関数として MakeStr 関数が存在します(逆は lstrcpy 関数を使えばよいです)。

```
Dim str1[12] As Byte
lstrcpy(str, ″Sinryow″)
lstrcat(str, ″ Game″)
Print MakeStr(str) ' 「Sinryow Game」と表示
```

5. 可変長の配列

単に Byte 型配列を文字列として扱うには，決定的な弱点があります。それは文字列の長さが固定長になることです。Byte 型に限らず，可変長の配列は以下のようにして扱います。

```
Dim str As BytePtr
str=malloc(8)
lstrcpy(str, "Sinryow")
Print MakeStr(str) ' 「Sinryow」と表示
Print str[0] ' 「Sinryow」の１文字目の文字コード（83）を表示

str=realloc(str, 12)
lstrcpy(str, "ActiveBasic")
Print MakeStr(str) ' 「ActiveBasic」と表示

free(str)
```

2 行目の malloc 関数から説明します。これはヒープ領域（メモリ中で特に動的に使う領域）から指定されたバイト数を使うことを宣言してそのバイト数を確保し（メモリ上のどの部分を使うかは自動的に決定されます），その領域の先頭のアドレスを返します。アドレスを返してくるわけですから，返り値はポインタ型変数で受け取らなければなりません。
この例だと「malloc(8)」としていますから，変数 str はあたかも「Dim str[7] As Byte」と宣言したかのように使用できます。

7 行目の realloc 関数は，一度 malloc（あるいは realloc）で既に確保されている領域を，再度バイト数を変えて確保し直す関数です。第1引数には前に確保した領域の先頭アドレスを与えます。この例だと「realloc(str, 12)」としていますから，最初の malloc で確保した領域を取り直して 12 バイトの領域を取得しています。
（注）realloc で領域を再確保した場合，必ずしも元の領域と同じ領域が使えるわけではないので注意して下さい。

最後の free 関数は，malloc や realloc で確保した領域を破棄します。こうしないとそのヒープ領域が後で（別のアプリケーションでも）使えないままになってしまうからです。

一応確認しておきますが，malloc・realloc で確保する領域は（要素数ではなく）バイト数で指定することに気を付けて下さい。
下にサンプルを載せます。

```
#prompt
Dim person As Long, weight As SinglePtr, sum As Single, i As Long, c As Single
weight=NULL ' "NULL"は，ポインタ変数に０が入っている（アドレス０を指している）ことを表す。
Do
    ' 人数を入力
    Print "体重の平均を求めます"
    Input "人数 >", person
    If weight=NULL Then
        weight=malloc(person*4) ' Single 型は４バイトなので，変数の数×４バイトが必要。
    Else
        weight=realloc(weight, person*4)
    End If

    ' 各人の体重を入力
    For i=0 To person-1
        Print i;
        Input "番目の人の体重>", weight[i]
    Next

    ' 平均を計算
    sum=0
    For i=0 To person-1
        sum=sum+weight[i]
    Next
    Print "平均"; sum/person
    Print "終了するにはＱキー，続ける場合は別のキーを押して下さい。"
Loop Until Input$(1)="Q"

free(weight)
End
```

6. ポインタ変数に対する計算

ポインタ変数に対して計算を行うこともできます。その場合は DWord 型と同じように計算されます。
しかしポインタはあくまでアドレスを格納しているわけですから，変に値をいじられると正常に動作しないことも考えられます。ポインタ変数に対する計算は，通常は足し算と引き算くらいです。

ポインタ変数に対する計算の例にあまりよいものが思い付かないので，やや高度な例になりますが lstrcpy 関数を自分で作ってみました(mystrcpy)。

```
Function mystrcpy(str1 As BytePtr, str2 As BytePtr) As BytePtr
    Do
        SetByte(str1, GetByte(str2))
        If GetByte(str2)=0 Then Exit Do
        str1=str1+1
        str2=str2+1
    Loop
    mystrcpy=str1
End Function

Dim S As BytePtr
Dim T[5] As Byte
S="BASIC"
mystrcpy(T, S)
Print MakeStr(T)
```

この例で，mystrcpy 関数の中でポインタ変数がどのように振る舞っているのかを解説します。
まず，mystrcpy を呼び出した時点でのアドレスおよびポインタ変数の値構成は以下のようになっています。

アドレス	str2						…	str1					
	S						…	T[0]	T[1]	T[2]	T[3]	T[4]	T[5]
値	'B'	'A'	'S'	'I'	'C'	0	…						

※シングルクォーテーションで囲った文字は，その文字コードを表すものとします。

まず，最初の「SetByte(str1, GetByte(str2))」を実行します。str2 の位置の値を str1 の位置にセットするわけですから，

アドレス	str2						…	str1					
	S						…	T[0]	T[1]	T[2]	T[3]	T[4]	T[5]
値	'B'	'A'	'S'	'I'	'C'	0	…	'B'					

となります。
str2 の指している値は 0 でないですから，次に進んで str1 と str2 の値をそれぞれ 1 ずつ増やすと，

アドレス		str2					…		str1				
	S						…	T[0]	T[1]	T[2]	T[3]	T[4]	T[5]
値	'B'	'A'	'S'	'I'	'C'	0	…	'B'					

と，str1 と str2 の位置が変わります。
ループを続けてもう一度「SetByte(str1, GetByte(str2))」を実行すると

アドレス		str2					…		str1				
	S						…	T[0]	T[1]	T[2]	T[3]	T[4]	T[5]
値	'B'	'A'	'S'	'I'	'C'	0	…	'B'	'A'				

となります。以下これを繰り返していくと，6回目の「SetByte(str1, GetByte(str2))」では

アドレス						str2	…						str1
	S						…	T[0]	T[1]	T[2]	T[3]	T[4]	T[5]
値	'B'	'A'	'S'	'I'	'C'	0	…	'B'	'A'	'S'	'I'	'C'	0

となります。ここで str2 の指している値が 0 になるのでループを抜けます。これで T[0]を先頭として6バイトの領域に"BASIC"の文字列をコピーすることに成功しました。実際には lstrcpy 関数もこれとほぼ同じことをしています。

補足('05.05.11 追加)

ActiveBasic ではポインタ変数への加減算をした場合アドレスの値はその数だけ移動しますが，C などではポインタ変数への加減算をした場合その「変数の数だけ」アドレスの値が変わります。

例えば ActiveBasic では

```
Dim data[9] As Long, i As Long
Dim ptr As DwordPtr
For i=0 To 9
    data[i]=i
Next
ptr=data
ptr=ptr+8 ' Long 型は 4 バイトなので，data[2]の位置を指すことになる
Print GetDWord(ptr)
```

とすれば，ptr は data[2]の位置を示し，値は「2」が表示されます。

一方 C で

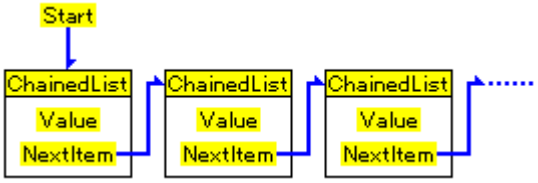
```
#include <stdio.h>

int main(void){
    int data[10], i;
    int* ptr;
    for(i=0; i<=9; i++){
        data[i]=i;
    }

    ptr=data;
    ptr=ptr+8; // ActiveBasic でのポインタの足し算とは意味が違う・・・
    printf("%d¥n", *ptr);

    return 0;
}
```

とすると，ポインタの足し算では「変数の数だけ」アドレスの値が変わるので，int（ActiveBasic の Long に相当）8個分，すなわち8×4＝32だけ ptr の値が変化します。結局 ptr は data[8]の位置を示すことになり，値は「8」が表示されます。

ActiveBasic と C の間でコードを移植する際は気をつけて下さい。(tak さん御指摘)

7. 応用 - 連結リスト

応用として，連結リストを扱います。これは下の図のように，構造体の中に構造体そのものへのポインタ(下図ではNextItem)を持たせてリストを連続させることです。




```
#prompt

'構造体定義
Type ChainedList
    Value As Single '型は適当に変えてOK
    NextItem As *ChainedList
End Type

Dim Start As *ChainedList, Last As *ChainedList
'第１のアイテムを準備
Start=malloc(8) 'Single型の４バイトと，*ChainedListの４バイトの合計８バイトが必要
                '※ポインタはすべて４バイト（32bitコンピュータの場合）
Last=Start
Last->Value=20
'第２のアイテムを準備
Last->NextItem=malloc(8)
Last=Last->NextItem
Last->Value=50
'第３のアイテムを準備
Last->NextItem=malloc(8)
Last=Last->NextItem
Last->Value=30
Last->NextItem=NULL 'NextItemにNULLを入れることで，連結リストの終端の
                    'アイテムであることを表すことが多い。
'領域の表示と開放（再帰プログラムを使用）
ViewValues(Start)

Sub ViewValues(ListItem As *ChainedList)
    If ListItem->NextItem<>NULL Then
        ViewValues(ListItem->NextItem)
    End If
    Print ListItem->Value
    free(ListItem)
End Sub
```

NextItemはポインタでしかないので，実際に次の値を入れる場合にはmallocで次のChainedList構造体を格納するための領域を確保する必要があります。
領域の開放には再帰呼び出しが便利です。再帰をしているため，連結リストの後ろにあるアイテムほど先に領域を開放されます(実行してみればわかります)。
現実にも応用範囲が広い構造体です。

～ 「ポインタとは何ぞや？」 終 ～