モデル記述のためのプログラミング 3

プログラミングの基礎（1）

07186　井原 智彦*

平成 14 年 5 月 24 日

1　手続き型プログラミングひとめぐり（2）

1.1　関数における引数の受け渡し

たとえば，2 つの値を交換する関数を考えてみる．以下のようにプログラムを書いたとしよう．

```
1  void Swap1( int a, int b ) {
2      int temp;
3      temp = a;
4      a = b;
5      b = temp;
6  }
```

そして別の関数から呼び出してみる．

```
1  void TestSwap() {
2      int value1, value2;
3      value1 = 3;
4      value2 = 4;
5      cout << "交換前の value1 は " << value1 << " 、"
6          << "value2 は " << value2 << " です。" << endl;
7      Swap1( value1, value2 );
8      cout << "交換後の value1 は " << value1 << " 、"
9          << "value2 は " << value2 << " です。" << endl;
10 }
```

以上の関数群を実行させるためには，たとえば，まとめて以下のように書く．関数の宣言や分割コンパイルなどに関しては後述.

```
1  #include <iostream>
2
```

* 東京大学大学院工学系研究科地球システム工学専攻博士課程，　E-mail ihara@globalenv.t.u-tokyo.ac.jp

```
 3  using namespace std;
 4
 5  void Swap1( int a, int b ) {
 6      int temp;
 7      temp = a;
 8      a = b;
 9      b = temp;
10  }
11
12  void TestSwap() {
13      int value1, value2;
14      value1 = 3;
15      value2 = 4;
16      cout << "交換前の value1 は " << value1 << " 、"
17          << "value2 は " << value2 << " です。" << endl;
18      Swap1( value1, value2 );
19      cout << "交換後の value1 は " << value1 << " 、"
20          << "value2 は " << value2 << " です。" << endl;
21  }
22
23  int main() {
24      TestSwap();
25      return 0;
26  }
```

しかし，値は交換されない．何故であろうか？　これは，Swap1(int a, int b) という関数に問題がある．

どういうことか説明すると，value1 の値を a にコピーして渡す，ということになる．a のスコープは当然 Swap1 関数内にとどまり，あくまでも値が交換されるのは，a や b であって，value1 や value2 ではない．そのため，呼び出し側には結果が反映されないのである．

一方，次のように書いてみる．

```
1  void Swap2( int& a, int& b ) {
2      int temp;
3      temp = a;
4      a = b;
5      b = temp;
6  }
```

ここで，int&型とは，int 型の参照型という意味である．つまり，a は値を持つのではなく，単に value1 の参照に過ぎない，ということになる．

参照渡しの場合，a は，value1 の参照となる．つまり，a の値を書き換えようとしても，a 自体は値を持っておらず単に value1 の参照に過ぎないので，その参照先の value1 の値が書き換わることになる．ゆえに，Swap2 内の a や b に対する一連の操作は，すべて value1 や value2 に対して操作をおこなっていることになる．

よって，TestSwap 関数から Swap2 関数を呼び出せば，value1 と value2 の値を交換することができる．

このように，参照渡しは便利であり，また値渡しに比べて高速（参照渡しは，値をコピーするのではなく，変数の参照を渡すだけなので）である．しかし，不用意に参照渡しをおこなうと，関数内でしか値を書き換えたくない場合に，呼び出し元まで書き換えてしまうことがある．また，関数内でのミスが呼び出し元に響くため，デバッグにも苦労することになる．そこで基本的には値渡しとし，Swap2 のように呼び出し元に反映させる必要がある場合，もしくは，変数のサイズがとても大きい場合[*1]に限って，参照渡しとするのが望ましい．これも，スコープをできるだけ限定しよう，という考えの 1 つといえる．

なお，参照渡しというのは，C++では比較的新しい概念である．従来は，次のようにポインタを使って同様のことを表現していた[*2]．ポインタ渡しの場合は，まず次のように Swap3 関数を作成する．

```
 1  void Swap3( int* a, int* b ) {
 2      int temp;
 3      temp = *a;
 4      *a = *b;
 5      *b = temp;
 6  }
```

そして，この関数を以下のようにして呼ぶ．

```
 1  void TestSwap2() {
 2      int value1, value2;
 3      value1 = 3;
 4      value2 = 4;
 5      cout << "交換前の value1 は " << value1 << " 、"
 6          << "value2 は " << value2 << " です。" << endl;
 7      Swap3( &value1, &value2 );
 8      cout << "交換後の value1 は " << value1 << " 、"
 9          << "value2 は " << value2 << " です。" << endl;
10  }
```

ここで，見慣れない演算子が 2 つほど見受けられる．*演算子と&演算子である．これらの演算子も含め，ポインタについて解説する．

まず，int*型とは，int 型へのポインタという意味である．上記で説明した参照（int&型）と異なるのは，

- 一番の違いは，参照は，実体と同じように使えるのに対し，ポインタはそうではないということである．参照型はあたかも実値が格納されているようにふるまうのに対し，ポインタ型には指し示した実体が存在するメモリ上のアドレスが格納されているに過ぎない．
- 参照は必ず実体が必要であるのに対し，ポインタは実体がなくとも存在しうる．つまり，参照は必ず定義されるのに対し，ポインタは宣言だけでも構わない．

である．

そのため，ポインタはそのままでは使えない．しかし，*演算子を使うと，実体に格納されている値を呼び

[*1] クラスや構造体．

[*2] 現在でもポインタ渡しの方が広く用いられている．過去から使われているという点が大きいが，加えて，参照渡しの場合，一見，その変数が，参照（int&型）なのか実体（int 型）なのか分からないので，書き間違えが発生し，ひいてはバグの原因になる，と嫌う人もいるためである．

出すことができる．これで，参照と同じように使える．一方，実体をポインタに渡すにあたっては，実体のアドレスを渡してやる必要がある．&演算子を用いれば，変数（実体）の存在する（メモリ上の）アドレスを呼び出すことができる．

ちなみに，参照は，ポインタを隠しているだけで，実際には，上記のような作業をコンパイラがやっているのである（つまりポインタを使おうが参照を使おうが同じ，ということ）．Visual Basic でも同様である（ただし，Visual Basic では C++のようないわゆるポインタは存在しない）．

なお，参照やポインタを使用した場合は，一連の TestSwap 関数で示したように，値の変更が可能である．しかし，場合によっては，値の変更をする必要がないこともある．その際，コーディングのミスで不要に変更してしまうのを防ぐ場合には，引数の前に，const を付加してやればよい．

```
1  void Swap4( const int& a, int& b ) {
2      int temp;
3      temp = a;
4      a = b;                                    // エラー!
5      b = temp;
6  }
```

```
1  void Swap5( const int* a, int* b ) {
2      int temp;
3      temp = *a;
4      *a = *b;                                  // エラー!
5      *b = temp;
6  }
```

■Visual Basic の場合　Visual Basic においても，全く同じである．値渡しをする Swap1 プロシージャを作成する．

```
1  Sub Swap1(ByVal intA As Integer, ByVal intB As Integer)
2      Dim intTemp As Integer
3      intTemp = intA
4      intA = intB
5      intB = intTemp
6  End Sub
```

そして別のプロシージャから呼び出してみる．引数宣言の頭にある ByVal というのは「値渡し」という意味である．

```
1  Sub TestSwap()
2      Dim intValue1 As Integer, intValue2 As Integer
3      intValue1 = 3
4      intValue2 = 4
5      Call MsgBox("交換前の intValue1 は " & CStr(intValue1) & " 、" _
6          & "intValue2 は " & CStr(intValue2) & " です。")
7      Call Swap1(intValue1, intValue2)
8      Call MsgBox("交換後の intValue1 は " & CStr(intValue1) & " 、" _
```

```
 9          & "intValue2は " & CStr(intValue2) & " です。")
10  End Sub
```

すると，予想通り，値は交換されない．次のように参照渡しをする Swap2 の場合は，値は交換される．

```
 1  Sub Swap2(ByRef intA As Integer, ByRef intB As Integer)
 2      Dim intTemp As Integer
 3      intTemp = intA
 4      intA = intB
 5      intB = intTemp
 6  End Sub
```

ここで，ByRef は「参照渡し」を意味する．ちなみに，Visual Basic では，ByVal もしくは ByRef というキーワードを省くと，自動的に，引数は参照渡しとなる．