ポインターの応用 (配列と文字列)

山本昌志*

2007 年 2 月 16 日

概 要

ポインターの基本的な使いかたを学習する．はじめに，ポインターの初期化と整数演算の意味を説明する．その後，配列との関係，文字列リテラルを説明する．

1 前回の復習と本日の内容

1.1 前回の復習

前回の講義では，まずはじめにコンピューターの簡単な仕組みとメモリーの概要を説明した．そこで，理解しておきべき内容は次のようなことである．

- プログラム (データと命令) は，メモリーの中に格納される．
- CPU はメモリーからデータと命令を読み込む．読み込んだ命令にしたがい，データを処理して，結果をメモリーに戻す．
- メモリーには，データの場所を示す整数のアドレスがある．諸君が使っているパソコンでは，アドレスは 32 ビットである．32 ビットなので，16 進数で表すと 8 桁である．
- メモリーのひとつのアドレスには，1 バイト (8 ビット，16 進数の 2 桁) の情報を格納することができる．

メモリーを理解した後，ポインターの基礎を学習した．そこでのおもな内容は，次のとおりである．

- ポインター変数は，アドレスを格納する変数である．
- ポインターの宣言には，型名とアスタリスク (*) を付ける．
- 変数のアドレスを取り出すには，変数名の前にアンパサンド (&) をつける．&はアドレス演算子である．`printf()` 関数でアドレスを表示させるときには，変換指定子`%p` を使う．
- ポインターが示しているデータの値を取り出すためには，ポインター変数の前にアスタリスク (*) をつける．*は間接参照演算子である．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

1.2 本日の学習内容

本日の学習の範囲は，教科書 [1] の p.276–287 である．学習のゴールは，以下のとおりである．

- ポインターを使うためには，初期化が必要である—ことが分かる．
- 可能なポインターの演算は，整数の加算と減算のみである．1 加算や減算はそのデータのバイト数分のアドレスが移動する．これらのポインターの演算が分かる．
- 配列とポインターの関係が分かる．
- 文字列リテラル (定数) の使い方が分かる．

2 ポインターの初期化

ポインターは，データオブジェクトの先頭アドレスを格納する変数である．その中にデータのアドレスを格納してはじめて，ポインターが意味のあるデータとなる．

これを，リスト 1 のプログラムを使って説明しよう．最初，ポインター `i_ptr` の値 (アドレス) は `0x90ee40` は，このプログラムでは全く意味のないものである．実行環境—パソコンや実行するタイミングなど—が変化すれば，このポインターが指し示しているデータオブジェクトの値 `0x57e58955` も変わってしまう．同じプログラムなのに，出力が変わるようでは役に立たない．

ポインターを使う場合，このプログラムの後半 (11 行目) に示しているように，使うアドレスを明示的に格納しなくてはならない．これを**ポインターの初期化**と呼び，ポインターを使う場合の基本的な約束である．これはプログラマーの仕事で，初期化を忘れてもコンパイラーはエラーメッセージを表示してくれない．

リスト 1: 初期化していないポインターと初期化後のポインター

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int i=0x12345678;
6    int *i_ptr;
7
8    printf("i_ptr に格納されているアドレス\t\t%p\n",i_ptr);
9    printf("i_ptr が示すデータオブジェクトの値\t%#x\n",*i_ptr);
10
11   i_ptr=&i;
12   printf("i のアドレス\t\t\t%p\n",&i);
13   printf("i_ptr に格納されているアドレス\t\t%p\n",i_ptr);
14   printf("i_ptr が示すデータオブジェクトの値\t%#x\n",*i_ptr);
15
16   *i_ptr=0xfedcba98;
17   printf("iの値\t\t\t\t%#x\n",i);
18
19   return 0;
20
21 }
```

実行結果

```
i_ptr に格納されているアドレス          0x90ee40
i_ptr が示すデータオブジェクトの値      0x57e58955
```

i のアドレス	0xbfacc0cc
i_ptr に格納されているアドレス	0xbfacc0cc
i_ptr が示すデータオブジェクトの値	0x12345678
i の値	0xfedcba98

3 ポインター演算

3.1 どのような演算ができるか

ポインターは，整数型の値の和と差の演算ができる．しかし，積と商の演算はできない．すなわち，リスト 2 のようにポインターに整数を足したり引いたりする場合，エラーも無くコンパイルできて，実行も可能である．それにたいして，リスト 3 に示すかけ算や割り算の演算はコンパイル時にエラーとなる．

リスト 2: ポインターと整数の和と差の演算．このプログラムは実行できる．

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int *p1, *p2, *p3, *p4;
6    int i1=111, i2=222;
7
8    p1=&i1;
9    p2=&i2;
10
11   p3=p1+3333;
12   p4=p1-4444;
13
14   return 0;
15 }
```

リスト 3: ポインターと整数の積と商の演算．このプログラムは，コンパイル時にエラーが発生する．

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int *p1, *p2, *p3, *p4;
6    int i1=111, i2=222;
7
8    p1=&i1;
9    p2=&i2;
10
11   p3=p1*3333;
12   p4=p1/4444;
13
14   return 0;
15 }
```

3.2 和や差の演算の意味

ポインターに整数を足したり，引いたりすることができることが分かった．これには，どういう意味があるのだろうか? リスト 4がそれに対する答えである．ポインターの演算—整数の加算と減算—は，データの型のバイト数分，ポインターがシフト (移動) する．プログラムの実行結果から分かるように，

- 文字型のポインターでは，整数の 1 加算はアドレスの値が 1 増える．
- 整数型のポインターでは，整数の 1 加算はアドレスの値が 4 増える．
- 倍精度実数型のポインターでは，整数の 1 加算はアドレスの値が 8 増える．

となっている．

ポインターに加算される整数は，ポインターが指し示すデータの移動量を表す．ひとつのアドレスには 1 バイトのデータが格納できる．そして，文字型のデータでは 1 バイト，整数型では 4 バイト，実数型では 8 バイトのメモリーである．これが，型に依存して，アドレスの変化の仕方が異なった理由である．

リスト 4: ポインターに整数を加算した場合のアドレスの変化

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    char *cp;
6    int *ip;
7    double *dp;
8    int i;
9
10   for(i=0; i<4; i++){
11     printf("%d %p\t%p\t%p\n",i, cp+i, ip+i, dp+i);
12   }
13
14   return 0;
15 }
```

実行結果

```
0 0xbff0eb9c    0xbff0eb08      0x8048416
1 0xbff0eb9d    0xbff0eb0c      0x804841e
2 0xbff0eb9e    0xbff0eb10      0x8048426
3 0xbff0eb9f    0xbff0eb14      0x804842e
```

4 配列

同じ型のデータが大量にある場合，配列を使うと便利である．配列名と自然数の添え字によりデータが指定できるので，大量のデータでも容易にアクセスできる．この配列とメモリー及びポインターの関係を述べる．

ここでの話で特に重要なことは，配列の添え字から，データが格納されているアドレスを割り出す方法である．

4.1 一次元配列

C 言語では，配列名はデータの先頭を表すポインターのように動作する．以前学習したように，ポインターの値に 1 加算すると，次のデータを示すようになる．したがって，配列名に 1 加算すると，次の配列の値を示すポインターになる．これについては，具体例を示した方が分かりやすいだろう．リスト 5 を使って説明する．

4 行 整数型のポインター p と要素数 3 の整数型の配列 a，整数型の変数 i の宣言
10 行 配列 a の先頭アドレスをポインター p へ代入
12 行 ポインター p のアドレスと，そこに格納されているアドレスを表示
15 行 配列のアドレスと値，a をポインターと考えた場合のアドレスと値，ポインター p のアドレスと値の表示

リスト 5: 1 次元配列とアドレス

```
1  #include <stdio.h>
2
3  int main(void){
4    int *p, a[3], i;
5
6    a[0]=11;
7    a[1]=22;
8    a[2]=33;
9
10   p=a;
11
12   printf("%p %p\n", &p, p);
13
14   for(i=0;i<3; i++){
15     printf("%p %d %p %d %p %d\n", &(a[i]), a[i], a+i, *(a+i), p+i, *(p+i));
16   }
17
18   return 0;
19 }
```

実行結果

```
0xbffff69c 0xbffff680
0xbffff680 11 0xbffff680 11 0xbffff680 11
0xbffff684 22 0xbffff684 22 0xbffff684 22
0xbffff688 33 0xbffff688 33 0xbffff688 33
```

このプログラムの実行直後のメモリーの様子を図 1 に示す．このメモリーの様子と実行結果から，次に示す 2 つのことが分かるだろう．

まずは，配列名は配列の先頭アドレスを示すポインターのように動作する．したがって，リスト 5 の 10 行目のように，左辺値として配列名を指定して，それをポインターに代入することができる．

次に分かることは，配列への要素のアクセスは，ポインターを使って表現できる．すなわち，`a[i]` は，`*(a+i)` と同じであることがわかる．事実，コンパイラーはこのようにしてメモリーにアクセスするように機械語に変換するのである．

図 1: プログラム実行後のメモリーの様子

- 配列名は配列の先頭を示すポインターのように動作する．
- 配列の要素へのアクセス a[i] は，*(a+i) と同じである．

4.2 多次元配列

C 言語の多次元配列について正確に述べようとすると，ここでの説明よりも，さらにもっと込み入った話がある．ますます混乱する者が多くなりそうなので，ここではコンパイラーの動作を考慮した細かい説明は避ける．興味のある者は，自分で学習せよ．

これまでの話で，一元のポインターは分かった．2 次元以上はどうなっているのだろうか?．同じようにプログラムを作成して調べてみるのが良いだろう．ここでは，配列の添え字からどのようにしてメモリーのアドレスの導出方法に興味がある．そのために，リスト 6 の 2 次元の配列を使ったプログラムを考える[1]．

このプログラムの実行結果と図 2 のメモリーの様子から，a[i][j] は配列名の先頭アドレス a に 3*i+j 加算したアドレスになることが分かる．

さらに，ここでも配列はポインターを用いて，

a[i][j] → *(a+i)[j] → *(*(a+i)+j)

となっていることが分かる．ここのところは分からなくても良い．

リスト 6: 1 次元配列とアドレス

```
1  #include <stdio.h>
2
3  int main(void){
4    int a[2][3], i,j;
5    int *p;
6
7    p=a;
8
9    printf("pointer p address %p  value %p\n", &p, p);
10
11   a[0][0]=0;   a[0][1]=1;   a[0][2]=2;
12   a[1][0]=10;  a[1][1]=11;  a[1][2]=12;
```

[1]このプログラムの 7 行目はコンパイラーが警告を出すが実行は可能である．

```
13
14    for(i=0; i<2; i++){
15      for(j=0; j<3; j++){
16        printf("%p %d %p %d %p %d\n",
17               &(a[i][j]), a[i][j], p+3*i+j, *(p+3*i+j), *(a+i)+j, *(*(a+i)+j));
18
19      }
20    }
21
22    return 0;
23  }
```

実行結果

```
pointer p address 0xbffff674  value 0xbffff680
0xbffff680 0 0xbffff680 0
0xbffff684 1 0xbffff684 1
0xbffff688 2 0xbffff688 2
0xbffff68c 10 0xbffff68c 10
0xbffff690 11 0xbffff690 11
0xbffff694 12 0xbffff694 12
```

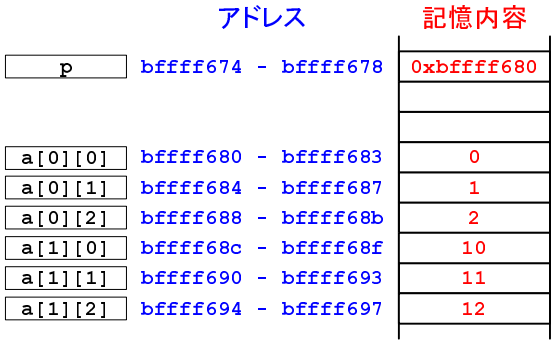

図 2: 2 次元配列を使ったプログラムの実行後のメモリーの様子

- `int a[10][20]` と配列を定義した場合を考える．このとき，`a[i][j]` のデータは，配列の先頭アドレス `a` に `20*i+j` 加算したアドレスに格納される．
- `int a[10][20][30]` と配列を定義した場合を考える．このとき，`a[i][j][k]` のデータは，配列の先頭アドレス `a` に `(20*30)*i+30*j+k` 加算したアドレスに格納される．
- 4 次元以上の配列でも同じ．

5　文字リテラル

最後にポインターのおもしろい使い方を示そう．おもしろいだけでなく，使い方によってはかなり便利な機能である．ここの部分は，教科書の p.285–287 に記述されている内容に対応する．

ダブルクォーテーション””で囲まれた文字列を文字型リテラル (文字列定数)[2]と呼ぶ．この文字型リテラルを使うと，その文字列がメモリーのどこかに格納されて，その先頭アドレスを返す．実際の例を，リスト 7 に示す．たぶん，プログラムを見れば，その結果は予想できるであろう．

リスト 7: 1 次元配列とアドレス

```
1  #include <stdio.h>
2
3  int main(void){
4
5    char *p;
6
7    p="Hello World !!";
8
9    printf("%s\n", p);
10
11   return 0;
12 }
```

実行結果

```
Hello World !!
```

鋭い学生は，配列を使わないで，文字列を取り扱っているところに気が付くであろう．文字列の講義では，文字列を扱うためには配列を使わなくてはならないと述べた．そして，任意の文字列を配列に代入するためには，`spfintf()` あるいは `strcpy()` 関数を用いる必要があり，代入演算子は使えないと述べたはずである．

しかし，ここでは配列を使わないし，代入演算子で文字列を代入している．このプログラムの 7 行目は，次のように動作するのである．

- ダブルクォーテーションで囲まれた文字列に文字の区切りを付加して Hello World !!\0 をメモリーのどこかに格納する．そして，その先頭アドレスを返している．
- 返された先頭アドレスは，ポインター p に代入している．

このようなことから，ポインター p は，配列名と同じ働きができるのである．従って，9 行目で文字列がディスプレイに表示できる．

ポインター p は，配列名と同じなので，

```
printf("%c",p[0]);
```

[2] リテラル情報はコンパイルするとき、実行ファイルにその値が直接埋め込まれる．

として，H の文字を表示することも可能である．

ただし，配列と異なりこの方法では，ポインターが指し示すアドレスの値を書き換えることは，大抵の場合，許されない．例えば，リスト 8 のように値を書き換えるプログラムではコンパイルはできるが，実行時に「セグメンテーション違反です」とエラーが発生する．リテラル (定数) は，書き換えが許されないアドレスにデータが格納されるからである．

リスト 8: ”Hello World !!”と表示させた後，”Hello Akita !!”と表示させるプログラム．これは実行時にエラーが発生する．

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5    char *p ;
6
7    p="Hello World !!";
8    printf("%s\n", p);
9    *(p+6)='A';
10   *(p+7)='k';
11   *(p+8)='i';
12   *(p+9)='t';
13   *(p+10)='a';
14   printf( "%s\n", p);
15
16   return 0 ;
17 }
```

実行結果

```
Hello World !!
セグメンテーション違反です
```

配列を使った方法だと書き換え可能である．したがって，データを書き換える可能性がある場合は，配列を使えばよい．書き換えないときは，ポインターを使った方が簡単である．

6 プログラム作成の練習

[練習 1] ポインターの値を 1,2,3,4 減算して，そのアドレスを調べよ．ポインターの型は文字型，整数型，倍精度実数型とする．

[練習 2] サイズが 100 の整数型の配列を用意して，1〜100 の整数を格納する．格納された整数をポインターを使って，表示せよ．

[練習 3] 文字列リテラルとポインターを使って，”Akita National College of Technology”と表示させよ．

[練習 4] 配列の添字とアドレスとの関係を調べるプログラムを作成せよ．そして，配列名をポインターに格納して，ポインターの加算と配列の添字の関係を調べよ．

7　課題

次の講義 (2 月 23 日) の AM8:45 までに，以下の課題をレポートとして提出すること．表紙等は，いつもの通り．表紙のタイトルは「ポインターの応用 (配列と文字列)」とすること．

[問 1] (復予) 教科書 [1]p.270–292 を 2 回読め．以下の問に答えよ．レポートには問いの答えとともに「2 回読んだ」と書け．

- 可能なポインター演算を示せ．
- 配列名は，何を表すか?
- 文字列リテラル (定数) のアドレスをポインターに格納する方法を示せ．

[問 2] (復) 本日配布したプリントを 2 回読め．レポートには「2 回読んだ」と書け．さらに，誤字・脱字，表現の悪いところ，間違いを指摘せよ．

[問 3] (復) サイズが 100 の整数型の配列を用意して，添字の順番に 1〜100 の整数を格納する．格納された整数をポインターを使って，表示せよ．ただし，表示は添字の逆の順序とすること．

[問 4] (予) 教科書 p.288 のリスト 8.12 が意図した通りに動作しない理由を示せ．

参考文献

[1] 内田智史監修, (株) システム計画研究所編. C 言語によるプログラミング　基礎編　第 2 版. (株) オーム社, 2006.