ポインター(配列とリテラル)

山本昌志*

2005年7月7日

1　本日の学習内容

前回の授業の授業に引き続き、ポインターについて学習する。前回は、ポインターを使って例として、関数のアドレス渡しについて学習した。

- 呼び出し側の関数は、自分の領域の変数のアドレスを引数として、呼び出した関数に渡す。
- 呼び出された側は、呼び出し側の変数のアドレスをポインターに格納する。
- このポインターに格納されたアドレスにより、呼び出された側は、呼び出しの領域のデータを操作できる。

前回の授業ではざっとこのようなことを学習したわけであるが、今回は配列と、文字リテラルについて学習する。

- 配列
- 文字リテラル

実際にポインターが本当に便利になるのは、もっと複雑なデータ構造を取り扱うときである。例えば、連結リストや木構造などである。これらについては、後期の学習範囲である。ポインターと言うものの概念が分からないと、後期の講義にはついていけないので、しっかり学習して欲しい。

2　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。そして、それを理解したならば、関数への配列の受け渡し方について考える。最終的には、配列を関数へ渡す方法を理解して欲しい。

*独立行政法人　秋田工業高等専門学校　電気情報工学科

2.1 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 1 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 1: 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 1 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り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。

2.2 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 2 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 2: 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
```

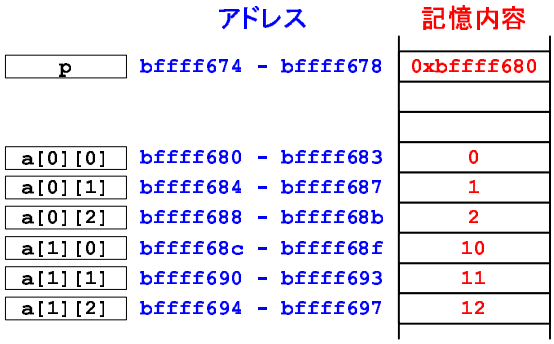

図 2: 2 次元配列を使ったプログラムの実行後のメモリーの様子

- `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。

2.3 関数への配列の受け渡し

ここでしつこく配列の添え字からメモリーのアドレスの出し方を示したのは、関数へ配列を渡す場合について説明したかったからである。

多次元配列の例でも分かるように、配列のデータへアクセスする場合、配列名と添え字の値と、配列のサイズが必要である。例えば、`int hoge[10][20][30]` と配列を定義した場合を考える。このとき、`hoge[i][j][k]` のデータにアクセスするためには、

- 配列の先頭アドレスを表す配列名 `a`
- 添え字の値 `i, j, k`
- 配列のサイズ 20 と 30

が必要である。配列のサイズの最初の値 10 は不要である。

このことから、関数に配列の全てを渡す場合には、配列名とサイズを渡す必要があることが分かる。ただし、配列名直後のサイズは不要である。配列の要素にアクセスする時には不要である。配列を引数にした関数は、リスト 3 のように書くことができるのである。

実行結果を見て分かるように配列の場合は、アドレス渡しとなっている。通常の変数は値渡しであるが、配列は特別な扱いを受けている。これは、通常の配列は非常に大きいので、値渡しでいちいちメモりーにコピーしていては時間がかかりすぎるからである。

リスト 3: 1 次元配列とアドレス

```
1  #include <stdio.h>
2  void double_array(int fuga[][20][30]);
3
4  /* ================================================== */
5  /*      main function                                 */
6  /* ================================================== */
7  int main(void){
8
9    int hoge[10][20][30];
10   int i, j, k;
11
12   for(i=0; i<10; i++){
13     for(j=0; j<20; j++){
14       for(k=0; k<30; k++){
15         hoge[i][j][k]=i+j+k;
16       }
17     }
18   }
19
20   double_array(hoge);
21
22   for(i=7; i<10; i++){
23     printf("hoge[%d][%d][%d]=%d\n", i, i, i, hoge[i][i][i]);
24   }
25
26   return 0;
27 }
28
29 /* ================================================== */
30 /*      function     double_array                     */
```

```
31 /* ========================================================== */
32 void double_array(int fuga[][20][30]){
33
34   int i, j, k;
35
36   for(i=0; i<10; i++){
37     for(j=0; j<20; j++){
38       for(k=0; k<30; k++){
39         fuga[i][j][k]=2*fuga[i][j][k];
40       }
41     }
42   }
43
44 }
```

実行結果

```
hoge[7][7][7]=42
hoge[8][8][8]=48
hoge[9][9][9]=54
```

- 配列を実引数にして関数に渡す場合、配列名を書けば良い。
- 受け取る側の仮引数は、配列名直後のサイズを指定する必要は無い。
- 配列はアドレス渡しである。

3 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。

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 4 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 4: 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。昨年の講義では、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]);
```

として、H の文字を表示することも可能である。

4 レポート

4.1 内容

以下のプログラムを作成して、レポートにまとめ提出すること。なお、このプログラムは提出日の次の日 (7 月 14 日) の授業の予習も兼ねている。したがって、正しく実行できなくても良く、手書きでプログラムを書くのも OK である。実行の確認は、来週の演習の時間に行う。

[**問 1**] 基礎

- 整数型変数 hoge と整数型のポインター fuga を使う。
- hoge には 123456 を代入、fuga には hoge を代入する。
- hoge のアドレスとその内容を表示する。
- fuga のアドレスとその内容と、ポインターが指し示す内容を表示する。

[**問 2**] 関数引数のアドレス渡し

- 整数型変数 a, b, c を使う。

– それぞれに、a=11, b=22, c=33 を代入する。
– これらの変数の値を 2 倍にする。ただし、アドレス渡しを使い、一つの関数で一度に 2 倍にすること。
– 2 倍した値をを表示する。

[**問 3**] 関数引数の配列の渡し

– 整数型の配列 array[11][22][33] をつかう。
– 配列の要素に、array[i][j][k]=i+j+k を代入する。
– 配列の要素の値を 3 倍にする。ただし、関数に配列のアドレスを渡し、一つの関数で 3 倍になるプログラムとすること。
– i=0〜10 までの、array[i][j][k] の値を表示する。

[**問 4**] リテラル

– 文字形のポインター pchar を使う。
– リテラルを使って、文字形のポインターに”Hello Akita”の先頭アドレスを代入する。
– 文字形のポインターを使って、代入された文を表示する。

4.2 レポート提出要領

提出方法は、次の通りとする。

期限	7 月 13 日 (水) PM 5:00
用紙	A4
提出場所	山本研究室の入口のポスト
表紙	表紙を 1 枚つけて、以下の項目を分かりやすく記述すること。 授業科目名「情報工学」 課題名「課題　ポインターの練習」 2E　学籍番号　氏名 提出日
内容	ソースプログラム (プリントアウトでも、手書きでも OK とする)