プログラミング Ⅰ

Report#7

提 出 日：２００７年７月19日
所　　属：工学部情報工学科
学籍番号：e075739A
氏　　名：津波古　正輝

【Report#7】

「問題２：配列＆ポインタ」より、解答確認のプログラムを作成し考察せよ。併せて、構造体，共有体について考察せよ。

ソースコード

```
/*  program      : prgrmng7a.c
    Student-Id   : 075739A
    Authar       : TSUHAKO Masaki
    Update       :2007/07/18(Thu)
    Comment      :解答確認          */
#include <stdio.h>
int main(){
printf("問２：配列とポインタの解答\n");

内容


}
```

内容を分けて行うことにする。

Q1:

```
{printf("Q1\n");
  /*Q1:&と*、ポインタ演算子**********************/
  int a=2,b=3,c=5,*p,*q;
  p=&b;
  q=&c;
  printf("a=%d,\t",a);
  printf("*p=%d,\t",*p);
  printf("*q=%d,\n",*q);
  printf("p=%d,\t",p);
  printf("q=%d,\n",q);
  printf("b のアドレス=%d,\t",&b);
  printf("c のアドレス=%d,\n",&c);
  a=*p+*q;
```

```
  printf("新しく変数を定義した後の a の値=%d\n",a);
}
```

出力結果：

```
Q1
a=2,    *p=3,   *q=5,
p=-1073743236,  q=-1073743240,
b のアドレス=-1073743236,        c のアドレス=-1073743240,
新しく変数を定義した後の a の値=8
```

変数宣言で a と b の箱を作り、ポインタ宣言で p と q の箱を作る。

p を出力してみると、*p=3 となっている。つまり、

*q も同様。

p を出力してみると、p=-1073743236 となる。(b のアドレス)

最初、a=2 と宣言していたが、その後、a=*p+*q と宣言。つまり、a=2 が a=*p+*q に上書きされてしまう。よって、a を出力してみると、a=8 となっている。

Q2:

```
{printf("Q2\n");
  /*Q2:アドレスと値の代入**********************/
    int a=2,*p;
    p=&a;
    *p=5;
    printf("a の値=%d\n",a);
    printf("p の値=%d\n",*p);

}
```

出力結果：

```
Q2
a の値=5
p の値=5
```

この結果を図に書いてみる。

変数宣言で、値『２』が入っている箱とポインタ宣言で p の箱を作る。

```
p=&a;
*p=5;
```

で、p に a のアドレスを入れる。そして、p の値を『５』にすると、間接的に、a の値も変わる。

Q3:

```
{
   /*Q3:配列とポインタ**************************/
  printf("Q3\n");
  int m[5]={1,5,2,4,3},*p;
  p=m;
  printf("m=%d\t",*m);
  printf("*(m+1)=%d\t",*(m+1));
  printf("p[0]=%d\t",p[0]);
  printf("p[2]=%d\n",p[2]);
  printf("m[0]=%d\t",m[0]);
  printf("m[1]=%d\t",m[1]);
  printf("m[0]=%d\t",m[0]);
  printf("m[2]=%d\n\n",m[2]);

  printf("m[0]のアドレス=%d\n",&m[0]);
  printf("m[1]のアドレス=%d\n",&m[1]);
  printf("m[2]のアドレス=%d\n",&m[2]);
  printf("m[3]のアドレス=%d\n",&m[3]);
  printf("m[4]のアドレス=%d\n\n",&m[4]);

  printf("m のアドレス=%d\n",&m);
  printf("&m+1 のアドレス=%d\n",&m+1);
  printf("&m+2 のアドレス=%d\n",&m+2);

}
```

出力結果：

```
Q3
m=1     *(m+1)=5        p[0]=1  p[2]=2
m[0]=1  m[1]=5  m[0]=1  m[2]=2
```

```
m[0]のアドレス=-1073743264
m[1]のアドレス=-1073743260
m[2]のアドレス=-1073743256
m[3]のアドレス=-1073743252
m[4]のアドレス=-1073743248

m のアドレス=-1073743264
&m+1 のアドレス=-1073743244
&m+2 のアドレス=-1073743224
```

出力結果から、*(m+1)は、m[1]と同じ意味である。

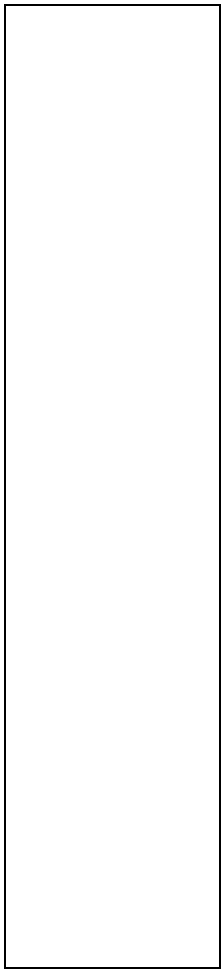
この結果を図に書いてみる。

アドレス

m

m+1

m+2



Q4:

```
{printf("Q4\n");
   /*Q4:int 型 1 次元配列*************************/
   int m[5]={10,20,40,50,30};
   printf("*m=%d\t",*m);
   printf("*(m+3)=%d\t",*(m+3));
   printf("*m+3=%d\t\t",*m+3);
   printf("*m+*(m+3)=%d\n",*m+*(m+3));
   printf("m[0]=%d\t\t",m[0]);
printf("m[3]=%d\t\t",m[3]);
printf("m[0]+3=%d\n",m[0]+3);

 printf("m[0]のアドレス=%d\t",&m[0]);
 printf("m[4]のアドレス=%d\n",&m[4]);
 printf("m のアドレス=%d\t\n",&m);

}
```

出力結果：

```
Q4
*m=10    *(m+3)=50        *m+3=13          *m+*(m+3)=60
m[0]=10          m[3]=50          m[0]+3=13
m[0]のアドレス=-1073743284       m[4]のアドレス=-1073743268
m のアドレス=-1073743284
```

*m は配列の最初(m[0])の値を指している。

*(m+3)は配列の三番目、つまり、m[0]から数えて三番目の値を指している。

m+3 は*m の値(m[0])に+3 することを意味している。

m+(m+3)は、*m の値に m[3]の値を加算することを意味してる。

Q5:

```
{printf("Q5\n");
   /*Q5:int 型 2 次元配列*************************/
   int d[4][3]={{1,2,3},{5,6,7},{4,6,8},{9,7,5}};
   printf("*d[2]=%d\t",*d[2]);
   printf("*(d[2]+2)=%d\t",*(d[2]+2));
   printf("*d[2]+2=%d\t",*d[2]+2);
   printf("**d=%d\n",**d);
   printf("*(*d+3)=%d\t",*(*d+3));
   printf("**d+6=%d\t",**d+6);
   printf("*(d[1]+2)=%d\t",*(d[1]+2));
   printf("**(d+2)=%d\n",**(d+2));

   printf("d[0][0]のアドレス=%d\t",&d[0][0]);
   printf("d[0][1]のアドレス=%d\n",&d[0][1]);
   printf("d[0][2]のアドレス=%d\t",&d[0][2]);
   printf("d[1][0]のアドレス=%d\n",&d[1][0]);
   printf("d[3][0]のアドレス=%d\n",&d[3][0]);

}
```

出力結果：

```
Q5
*d[2]=4 *(d[2]+2)=8     *d[2]+2=6       **d=1
*(*d+3)=5       **d+6=7 *(d[1]+2)=7     **(d+2)=9
d[0][0]のアドレス=-1073743388   d[0][1]のアドレス=-1073743384
d[0][2]のアドレス=-1073743380   d[1][0]のアドレス=-1073743376
d[3][0]のアドレス=-1073743352
```

イメージ図

*d[2]は、図でいう d[2][0]つまり、2 行目の 0 列を指している

***(d[2]+2)は、図でいう d[2][2]つまり、2 行目の 2 列目を指している。**

*d[2]+2 は、d[2][0]の値に 2 を加算するという意味である。

**d は、分かりやすく書くと、*d[0]という意味であり、つまり、d[0][0]を指している。

*(*d+3)は、d[0][0]から数えて三番目、つまり d[1][0]を指している。

**d+6 は、*d[0]の値に 6 を加算するという意味である。

*(d[1]+2)は、d[1][0]を、図でいう列方向に 2 ずらすという意味で、結果、d[1][2]を指す。

**(d+2)=4 は、d[0][0]を図でいう行方向に 2 ずらすという意味で、結果、d[0][2]を指す。

Q6

```
{printf("Q6\n");
  /*Q6:ポインタと文字列*************************/
  char *str="abcdefg",*p;
  p=str+3;
  printf("p=%s\n",p);
  printf("str のアドレス=%d\n",&str);
  printf("str(b の格納場所)のアドレス=%d\n",&str+1);
  printf("str(c の格納場所)のアドレス=%d\n",&str+2);
}
```

出力結果：

```
Q6
p=defg
str のアドレス=-1073743288
str(b の格納場所)のアドレス=-1073743284
str(c の格納場所)のアドレス=-1073743280
```

p=str+3 は、str[0]のアドレスに 3 を足して、str[3]からという意味。よって、p の中には、三番目からしか入っていない(0 番目から数えてなので、実質 4 番目から)

Q7:

```
{printf("Q7\n");
  /*Q7:配列、ポインタと文字列*******************71*/
  char m[]="abcdefghi";
  char *p,*q;
  p="jklmnopq";
  printf("*p=%c\t",*p);
  printf("*(p+2)=%c\n",*(p+2));

  p=&m[0]; q=m;
  printf("*m=%c\t",*m);
  printf("*(p+1)=%c\t",*(p+1));
  printf("*q+2=%c\n",*q+2);

  p=&m[3];
  printf("*p=%c\t",*p);
  printf("*(m+4)=%c\t",*(m+4));
  printf("*m+5=%c\n\n",*m+5);
```

```
    printf("p(j の格納場所)のアドレス=%d\t",&p);
    printf("p+1(k の格納場所)のアドレス=%d\n",&p+1);
    printf("p+2(l の格納場所)のアドレス=%d\t",&p+2);
    printf("p+3(m の格納場所)のアドレス=%d\n",&p+3);


  }
```

出力結果：

```
  Q7
  *p=j    *(p+2)=l
  *m=a    *(p+1)=b        *q+2=c
  *p=d    *(m+4)=e        *m+5=f

  p(j の格納場所)のアドレス=-1073743304     p+1(k の格納場所)のアドレス=-1073743300
  p+2(l の格納場所)のアドレス=-1073743296   p+3(m の格納場所)のアドレス=-1073743292
```

"jklmnopq"は、p の箱に 1 つ 1 つ格納されているのか、まとまって格納されているのか。

p の中に格納されている”’jklmnopq”のアドレスを見れば分かる。

j、k、l、m のアドレスは全て違う。ということは、別々に格納されているということだ。

*p は、p[0]を指している。よって、p にある”jklmnopq”すべてを指すのではなく、’j’一文字だけさす。

p=&m[0]と新たに宣言した後、p は m[]=”abcdefghi”の三番目、つまり’d’から代入されている。よって、*p は d と出力する。

m+5 は、*m が’a’を表しており、ASCII コードでいう 97 だ。その数字に+5 のなで、102 となる。それを、文字になおすと、f になる。よって、*m+5=f(102)となる。

Q8:

```
{printf("Q8\n");
  /*Q8:ポインタ、、文字置換**********************89*/
char *p,m[]="abcde";
p=m;
*(p+2)='x';
printf("p=%s\n",p);
*(p+3)='z';
printf("新しい定義も加えた p=%s\n",p);
}
```

出力結果：

```
Q8
p=abxde
新しい定義も加えた p=abxze
```

(p+2)とは c のこと。それを、*(p+2)='x'と宣言しているので、c が x に上書きされてしまう。

よって、『p=abxze』と出力される。

また、*(p+3)='z'と宣言しているので、上と同様に d が z に上書きされる。

よって、『p=abxze』と出力される。

Q9:

```
{printf("Q9\n");
  /*Q9:ポインタ配列と文字列*/
  char *q[]={"abcd","12345","ABCDEFG","987"};
  printf("*q[2]=%c\n",*q[2]);
  printf("q[3][2]=%c\n",q[3][2]);
  printf("*(q[2]+2)=%c\n",*(q[2]+2));
  printf("*(*(q+3)+2)=%c\n",*(*(q+3)+2));
  printf("**(q+1)=%c\n\n",*(q+1));
  printf("q(a の格納場所)のアドレス=%d\n",&q[0][0]);
  printf("q[0][1](b の格納場所)のアドレス=%d\n",&q[0][1]);
  printf("q[0][2](c の格納場所)のアドレス=%d\n",&q[0][2]);
  printf("q[0][3](d の格納場所)のアドレス=%d\n",&q[0][3]);
  printf("q[0][4](何もない)のアドレス=%d\n",&q[0][4]);
  printf("q[1][0](1 の格納場所)のアドレス=%d\n",&q[1][0]);
}
```

出力結果：

```
Q9
*q[2]=A
q[3][2]=7
*(q[2]+2)=C
*(*(q+3)+2)=7
**(q+1)=

q(a の格納場所)のアドレス=11280
q[0][1](b の格納場所)のアドレス=11281
q[0][2](c の格納場所)のアドレス=11282
q[0][3](d の格納場所)のアドレス=11283
q[0][4](何もない)のアドレス=11284
q[1][0](1 の格納場所)のアドレス=11288
```

*(q[2]+2)は、q[2][0]を列方向に 2 ずらすという意味である。よって、C を指す。

((q+3)+2)は、q[0][0]を、行方向に 3、列方向に 2 ずらすという意味で、q[3][2]を指す。

q(a の格納場所)のアドレス=11280

q[0][1](b の格納場所)のアドレス=11281

q[0][2](c の格納場所)のアドレス=11282

q[0][3](d の格納場所)のアドレス=11283

q[0][4](何もない)のアドレス=11284

q[1][0](1 の格納場所)のアドレス=11288

上の実行結果より、

a	b	c	d				
1	2	3	4	5			
A	B	C	D	E	F	G	
9	8	7					

Q10:

```
{printf("Q10\n");
  /*Q10:配列とポインタ*/
  char m[6]={'a','b','c','d','e','f'};
  char mm[2][3]={'a','b','c','d','e','f'};
  printf("m[4]=%c\n",m[4]);
  printf("*(m+2)=%c\n",*(m+2));
  printf("mm[0][0]=%c\n",mm[0][0]);
  printf("mm[1][1]=%c\n",mm[1][1]);
  printf("mm[0][2]=%c\n",mm[0][2]);
```

```
    printf("1 次元配列 m[]の先頭アドレスの表示方法(結果はすべて同じ)\n");
    printf("&m で表す\t%d(10 進)  %x(16 進)  %o(8 進)\n",&m,&m,&m);
    printf("&m[0]で表す\t%d(10 進)  %x(16 進)  %o(8 進)\n",&m[0],&m[0],&m[0]);
    printf("m で表す\t\t%d(10 進)  %x(16 進)  %o(8 進)\n",m,m,m);

    printf("1 次元配列 m[]の先頭の値の表示方法  m[0]→%c\n",m[0]);
    printf("1 次元配列 m[]の先頭の値の表示方法    *m→%c\n",*m);
    printf("Q45～Q48=2 次元配列 mm[][]の先頭アドレス\n");
    printf("\t%d(10 進)  %x(16 進)  %o(8 進)\n",&mm,&mm,&mm);
    printf("\t%d(10 進)  %x(16 進)  %o(8 進)\n",&mm[0][0],&mm[0][0],&mm[0][0]);
    printf("\t%d(10 進)  %x(16 進)  %o(8 進)\n",mm,mm,mm);
    printf("\t%d(10 進)  %x(16 進)  %o(8 進)\n",*mm,*mm,*mm);
    printf("\t%d(10 進)  %x(16 進)  %o(8 進)\n",mm[0],mm[0],mm[0]);

    printf("Q49～Q51=2 次元配列 mm[][]の先頭の値\t%c\n",**mm);
    printf("Q49～Q51=2 次元配列 mm[][]の先頭の値\t%c\n",mm[0][0]);
    printf("Q49～Q51=2 次元配列 mm[][]の先頭の値\t%c\n",*mm[0]);

    printf("Q52～Q55=2 次元配列[2][3]の 4 番目の値\t%c\n",mm[1][0]);
    printf("Q52～Q55=2 次元配列[2][3]の 4 番目の値\t%c\n",*mm[1]);
    printf("Q52～Q55=2 次元配列[2][3]の 4 番目の値\t%c\n",*(*mm+3));
    printf("Q52～Q55=2 次元配列[2][3]の 4 番目の値\t%c\n",**(mm+1));
}
```

出力結果：

```
Q10
m[4]=e
*(m+2)=c
mm[0][0]=a
mm[1][1]=e
```

```
mm[0][2]=c
1 次元配列 m[]の先頭アドレスの表示方法(結果はすべて同じ)
&m で表す          -1073743334(10 進)  bffffa1a(16 進)  27777775032(8 進)
&m[0]で表す       -1073743334(10 進)  bffffa1a(16 進)  27777775032(8 進)
m で表す           -1073743334(10 進)  bffffa1a(16 進)  27777775032(8 進)
1 次元配列 m[]の先頭の値の表示方法  m[0]→a
1 次元配列 m[]の先頭の値の表示方法     *m→a
Q45～Q48=2 次元配列 mm[][]の先頭アドレス
        -1073743340(10 進)  bffffa14(16 進)  27777775024(8 進)
        -1073743340(10 進)  bffffa14(16 進)  27777775024(8 進)
        -1073743340(10 進)  bffffa14(16 進)  27777775024(8 進)
        -1073743340(10 進)  bffffa14(16 進)  27777775024(8 進)
        -1073743340(10 進)  bffffa14(16 進)  27777775024(8 進)
Q49～Q51=2 次元配列 mm[][]の先頭の値       a
Q49～Q51=2 次元配列 mm[][]の先頭の値       a
Q49～Q51=2 次元配列 mm[][]の先頭の値       a
Q52～Q55=2 次元配列[2][3]の 4 番目の値      d
Q52～Q55=2 次元配列[2][3]の 4 番目の値      d
Q52～Q55=2 次元配列[2][3]の 4 番目の値      d
Q52～Q55=2 次元配列[2][3]の 4 番目の値      d
```

この実行結果より、次のことが分かった。

1 次元配列の先頭のアドレスの求め方　　&X[0]、　X、　&X　(X は、宣言する名前)

1 次元配列の先頭のアドレスの値の求め方　X[0]、　*X

2 次元配列の先頭のアドレスの求め方　　&XX[0][0]、XX[0]、XX、*XX、&XX

2 次元配列の先頭の値の求め方　　XX[0][0]、 **XX[0]**、**XX

構造体について：

　構造体とは、いろいろな種類の互いに関連するデータをまとめて１つのかたまりにしたもの。例えば、名前、性別、年齢、身長、体重というデータを１塊にして、人という構造体を作ることができる。

サンプルプログラム

```
 1 /*
 2   Program    : struct.c
 3    Comment    : 構造体による閏年判断
 4 */
 5
 6 #include <stdio.h>
 7 /*構造体の型枠の宣言　テンプレート*/
 8 struct smp{
 9   int   day;   /*日*/
10  int   month; /*月*/
11  int   year;  /*年*/
12 }date,*pdate;
13
14
15 isleap1(struct smp d){
16  int r4,r100,r400;
17
18   r4   = d.year % 4;
19   r100 = d.year % 100;
20   r400 = d.year % 400;
21
22  return ( ((r4 == 0) && (r100 != 0)) || (r400 == 0) );
23  }
24
25 isleap2(struct smp *d){
26  int r4,r100,r400;
```

```
27
28   r4   = d->year % 4;
29   r100 = d->year % 100;
30   r400 = d->year % 400;
31
32   return ( ((r4 == 0) && (r100 != 0)) || (r400 == 0) );
33 }
34
35 main(){
36   date.day   = 24;
37   date.month = 2;
38   date.year  = 1900;
39
40   printf("%4d 年は閏年で%s\n",date.year,          /*構造体の参照*/
42           (isleap1(date) != 0)? "す。":"ない。");
43   printf("%x %x %x\n",&date.day,&date.month,&date.year);
44
45   printf("%4d 年は閏年で%s\n",date.year,
46           (isleap2(&date) != 0)? "す。":"ない。");
47   printf("%x %x %x\n",&date.day,&date.month,&date.year);
48 }
```

出力結果：

```
1900 年は閏年でない。
2060 2064 2068
1900 年は閏年でない。
2060 2064 2068
```

考察：

8〜12 行目で構造体の型の宣言をおこなっている。

```
struct smp{
  int   day;   /*日*/
```

```
  int   month; /*月*/
  int   year;  /*年*/
}date,*pdate;
```

構造体のタグ名は、smp

メンバは整数型の int が 3 つで、day、month、year である。

構造体の型の宣言を行った後に、同時にそれを型名 date と*pdate として定義(12 行目)

18～20 行目で構造体型変数の中にあるメンバを参照している。

```
  r4   = d.year % 4;
  r100 = d.year % 100;
  r400 = d.year % 400;
```

メンバを参照するには、変数の後に“.”を付け、その後に、メンバ名を付ける。

ここでは、メンバ名『year』、変数名『d』を参照している。(15 行目で d は宣言)

参照した d.year をそれぞれ計算した後、r4、r100、r400 という変数に代入している。

22～23 行目で値を返す。

```
  return ( ((r4 == 0) && (r100 != 0)) || (r400 == 0) );
}
```

28～30 行目で構造体へのポインタを参照している。

```
  r4   = d->year % 4;
  r100 = d->year % 100;
  r400 = d->year % 400;
```

ポインタを参照するには、変数の後に“->”を付け、その後に、メンバ名を付ける。

ここでは、メンバ名『year』、変数名『d』のポインタを参照している。

35～38 行目で構造体の int 型のそれぞれのメンバに値を代入している。

```
main(){
  date.day   = 24;
  date.month = 2;
  date.year  = 1900;
```

40〜42 行目は、出力である。

```
printf("%4d 年は閏年で%s\n",date.year,          /*構造体の参照*/
       (isleap1(date) != 0)? "す。":"ない。");
```

38行目でdate.year=1900と宣言してあるので、『1900年は閏年で』と出力。

”す。”と”ない。”は「:」て区切られている。

この:を使って、閏年てあるか無いかと区別している。入力された年が真であるなら、”す。”を、偽なら(閏年てないなら)”ない。”を出力する。

45〜46行目も出力で、上と変わっていないように見えるが、isleap2になっている。

```
printf("%4d年は閏年で%s\n",date.year,
       (isleap2(&date) != 0)? "す。":"ない。");
```

このisleap2てはポインタを使っており、「->」を使ってdateをyearに取り込んている。

```
printf("%x %x %x\n",&date.day,&date.month,&date.year);
```

は、アドレスの表示。

共有体について：

サンプルプログラム

```
/*
  Program    : union.c
  Comment    : 共用体
*/

#include <stdio.h>

union smp{
  char  b08;
  short b16;
  int   b32;
};

main(){
  union smp var;

  var.b08=2;
  puts("---------------");
  puts("var.b08=2");
  printf("var.b08=0x      %02x=%d\n",var.b08,var.b08);
  printf("var.b16=0x    %04x=%d\n",var.b16,var.b16);
  printf("var.b32=0x%08x=%d\n",var.b32,var.b32);
  printf("var ADDRESS\n");
  printf("var.b08=0x%x +=%%0x%x\n",&var.b08,&var.b08+1);
  printf("var.b16=0x%x +=%%0x%x\n",&var.b16,&var.b16+1);
  printf("var.b32=0x%x +=%%0x%x\n",&var.b32,&var.b32+1);
}
```

出力結果：

```
---------------
var.b08=2
```

```
var.b08=0x      02=2
var.b16=0x    fffffb02=-1278
var.b32=0xbffffb02=-1073743102
var ADDRESS
var.b08=0xbffffa9c +=%0xbffffa9d
var.b16=0xbffffa9c +=%0xbffffa9e
var.b32=0xbffffa9c +=%0xbffffaa0
```

共有体（union は構造体とよく似ている。定義の仕方、メンバの参照方法も構造体と同じである。では、なにが違うのか

それはメンバが同じメモリを共有しているということだ。

上の実行結果を見てみると、アドレスでb08もb16もb32もすべて始まりのアドレスは同じということだ。b02のアドレスをb16とb32が、b16のアドレスをb32が共有している、ということになる。

感想：

今回は、本当に時間がなかった。しかも、難しかった～～～～～～～。共有体について、まだ不十分なところがあるので、これからもがんばります！！！！