プログラミング I

REPORT#7

提出日	:	2012年 7月19日　木曜日
所属	:	琉球大学工学部情報工学科
学籍番号	:	125722G
氏名	:	知念 辰明

1.各問題ごとにブロック文を用いて一つの解答確認プログラムとして作成し考察せよ。

q1:変数 v のアドレスを求める式を示せ。

高水準コード,プログラム説明から Q1 ブロックまで▼

```
1   /*
2     program   :proto.c
3     student-ID:125722G
4     author    :Tatsunori CHINEN
5     date      :07/17/2012 (SAT)
6     comment   :show answer!
7    */
8   #include <stdio.h>
9
10  int main(){
11
12      {   //====Q1
13          puts(" Q1.変数vのアドレスを求める式を示せ。");
14          int v;
15          puts("===>変数vのアドレスを求める式は&vである。");
16          printf("    &v = %8x\n\n",&v);
17      }
```

該当箇所の実行結果▼

```
 Q1.変数vのアドレスを求める式を示せ。
===>変数vのアドレスを求める式は&vである。
    &v = 6db18b44
```

考察▼

・int 型変数 v を宣言し,printf()関数への変数名にアドレス演算子"&"を付け出力することで変数 v のアドレスを求めた。
・よって,求める式は"&v"である。
・アドレス演算子"&"を付けなかった場合,変数 v に格納されている値が出力される。

q2:1 次元配列 m の 0 番目から始まる 5 番目の要素のアドレスを求める式を 2 つ示せ。

高水準コード,Q2 ブロック▼

```
18
19      {   //====Q2
20          puts(" Q2.1次元配列mの0番目から始まる5番目の要素のアドレスを求める式を2つ示せ。");
21          int m[6] = {0,0,0,0,0,1};
22          puts("===>1次元配列mの0番目から始まる5番目の要素のアドレスを求める式は\n     &m[5] と m[5] である。");
23          printf("    &m[5] = %8x, m+5    = %8x\n",&m[5],m+5);
24          printf("     m[5] = %8d, *(m+5) = %8d\n\n",m[5],*(m+5));
25      }
```

該当箇所の実行結果▼

```
 Q2.1次元配列mの0番目から始まる5番目の要素のアドレスを求める式を2つ示せ。
===>1次元配列mの0番目から始まる5番目の要素のアドレスを求める式は
     &m[5] と m[5] である。
    &m[5] = 6db18b40, m+5    = 6db18b40
     m[5] =        1, *(m+5) =        1
```

考察▼

・配列の中身を表す式"m[5]"にアドレス演算子"&"を付けたものと,配列の先頭アドレスである式"m"に配列要素の 5 番目を示すようにアドレスを変化させる"+5"を付けた。
・よって,求めた式は "&m[5]" と "m+5" である。

q3:1 次元配列 m の先頭アドレスを求める式を 2 つ示せ。

高水準コード,Q3 ブロック▼

```
26
27      {   //====Q3
28          puts(" Q3.1次元配列mの先頭アドレスを求める式を、2つ示せ。");
29          int m[6] = {1,0,0,0,0,0};
30          puts("===>1次元配列mの先頭アドレスを求める式は, &m と m である。");
31          printf("    &m = %08x,  m = %8x\n",&m,m);
32          printf("    *m = %8d\n\n",*m);
33      }
```

該当箇所の実行結果▼

```
 Q3.1次元配列mの先頭アドレスを求める式を、2つ示せ。
===>1次元配列mの先頭アドレスを求める式は, &m と m である。
    &m = 6db18b14,  m = 6db18b14
    *m =        1
```

考察▼

・1 次元配列の場合,配列名に何も付けなかった場合は配列の先頭を示すアドレスとなる。
・また,配列名にアドレス演算子"&"を付けた場合であっても配列の先頭を示すアドレスとなる。

q4:2 次元配列 d の先頭アドレスを求める式を 3 つ示せ。

高水準コード,Q4 ブロック▼

```
34
35      {   //====Q4
36          puts(" Q4.2次元配列dの先頭アドレスを求める式を、3つ示せ。");
37          int d[5][2];
38          d[0][0] = 1;
39          printf("===>2次元配列dの先頭アドレスを求める式は、 *d と &d[0][0] d[0]である。\n");
40          printf("    *d = %8x, &d[0][0] = %8x,  d[0] = %8x\n",*d,&d[0][0],d[0]);
41          printf("   **d = %8d,  d[0][0] = %8d, *d[0] = %8d\n\n",**d,d[0][0],*d[0]);
42      }
```

該当箇所の実行結果▼

```
 Q4.2次元配列dの先頭アドレスを求める式を、3つ示せ。
===>2次元配列dの先頭アドレスを求める式は、 *d と &d[0][0] d[0]である。
    *d = 6db18aec, &d[0][0] = 6db18aec,  d[0] = 6db18aec
   **d =        1,  d[0][0] =        1, *d[0] =        1
```

考察▼

・2 次元配列の場合,1 次元配列とは違い配列名だけでは配列の先頭アドレスを示さない。逆参照演算子"*"が付くことで先頭アドレスを示すようになる。
・"d[0][0]"だけでは 0 行 0 列に格納されている値"1"を示す式だが,アドレス演算子"&"が付くことで 0 行 0 列,つまり配列の先頭を示すアドレスを表す式となる。
・"d[0]"について,行を指定することで 1 次元配列のときと同様に演算子を付けずに先頭アドレスを表すことができた。

q5:変数 a の値を示せ。

高水準コード,Q5 ブロック▼

```
43
44      {   //====Q5
45          puts(" Q5.変数aの値を示せ。");
46          int a=2, b=3, c=5, *p, *q;
47          p = &b;
48          q = &c;
49          a = *p + *q;
50          puts("===>変数aの値は8である。");
51          printf("    a = %d\n\n",a);
52      }
```

該当箇所の実行結果▼

```
 Q5.変数aの値を示せ。
===>変数aの値は8である。
    a = 8
```

考察▼

・整数型ポインタ変数"p"および"q"は,逆参照演算子"*"によってそれぞれが表すアドレスに格納されている値を示す。
・よって,49 行目の演算はポインタ"p"および"q"が表す値,つまり"3"と"5"の足し算となっている。

q6:変数 a の値を示せ。

高水準コード,Q6 ブロック▼

```
53
54      {   //====Q6
55          puts(" Q6.変数aの値を示せ。");
56          int a=2, *p;
57          p = &a;
58          *p = 5;
59          puts("===>変数aの値は5である。");
60          printf("    a = %d\n\n",a);
61      }
```

該当箇所の実行結果▼

```
 Q6.変数aの値を示せ。
===>変数aの値は5である。
    a = 5
```

考察▼

・57 行目で整数型ポインタ変数"p"に"a"のアドレスが入力され,逆参照演算子を使った"*p"が"a"を表すようになっている。
・よって,58 行目の演算は"*p"が示す値の格納場所"a"に"5"を入力している。

q7:*p,*q,**q の値を示せ。

高水準コード,Q7 ブロック▼

```
62
63      {   //====Q7
64          puts(" Q7.*p,*q,**qの値を示せ。");
65          int a=200,b=300;
66          int *p,**q,*qx;
67          puts("    アドレス&aがアドレス100の,アドレス&bがアドレス200の代わりである。");
68          p = &a;
69          qx = &b;
70          q = &qx;
71          puts("===>*pの値は200, *qの値は200, **qの値は300を示す。");
72          puts("    ただし,*qの値はアドレス200に代わり,アドレス&bが示されている。");
73          printf("    *p = %d, *q = %x, **q = %d \n\n",*p,*q,**q);
74      }
```

該当箇所の実行結果▼

```
 Q7.*p,*q,**qの値を示せ。
    アドレス&aがアドレス100の,アドレス&bがアドレス200の代わりである。
===>*pの値は200, *qの値は200, **qの値は300を示す。
    ただし,*qの値はアドレス200に代わり,アドレス&bが示されている。
    *p = 200, *q = 6db18ab8, **q = 300
```

考察▼

- int 型変数 a の値を"200",b の値を"300"とし,これらのアドレスを整数型ポインタ変数"p","q"および"qx"に入力した。
- 問題にない新たなポインタ"qx"については,2 重ポインタ"**p"をそのまま用いた際実行時に"segmentation fault"というエラーが起こることを防ぐために宣言,使用している。
- a のアドレスを入力されたポインタ変数 p は,逆参照演算子を用いて値を参照すると a の値,つまり"200"を値とする。
- ポインタ変数"qx"のアドレスを"200"とし,qx に格納するアドレスとして"b"のアドレスを入力することで問題のような状況を再現している。
- 逆参照演算子を 1 つだけ付けた"*q"はポインタ変数"qx"のアドレス,つまり"200"を示し逆参照演算子を 2 つ付けた"**q"は更にポインタ変数"qx"に格納されているアドレス(&b)にある値"300"を参照している。

q8:1 次元配列 m において, m[k] と *(m+k)はどのような値か述べよ。

高水準コード,Q8 ブロック▼

```
75
76      {   //====Q8
77          puts(" Q8.1次元配列mにおいて, m[k] と *(m+k)はどのような値か述べよ。");
78          int m[6]={0,1,2,3,4,5};
79          puts("===>kを3とすると,m[k], *(m+k)のいずれも値3を示す。");
80          printf("    m[k] = %d, *(m+k) = %d\n\n",m[3],*(m+3));
81      }
```

該当箇所の実行結果▼

```
 Q8.1次元配列mにおいて, m[k] と *(m+k)はどのような値か述べよ。
===>kを3とすると,m[k], *(m+k)のいずれも値3を示す。
    m[k] = 3, *(m+k) = 3
```

考察▼

- k を 3 としたときの,m[k]と*(m+k)について観察した。
- 式"m[k]"は配列の 0 番目から数えて k 番目の値を示し,式"*(m+k)"は先頭アドレス"m"を"k"分ずらしたアドレスに格納されている値を表す。
- よってこの場合,どちらの式も同じ値を表している。

q9:整数型ポインタ変数 p において,p+2 は p の値を何バイト増加させた値か述べよ。

高水準コード,Q9 ブロック▼

```
82
83      {   //====Q9
84          puts(" Q9.整数型ポインタ変数pにおいて,p+2はpの値を何バイト増加させた値か述べよ。");
85          int *p;
86          printf("    p = %x, p+2 = %x\n\n",p,p+2);
87      }
```

該当箇所の実行結果▼

```
 Q9.整数型ポインタ変数pにおいて,p+2はpの値を何バイト増加させた値か述べよ。
    p = 0, p+2 = 8
```

考察▼

- このブロックでは,整数型ポインタ変数"*p"を宣言しその状態で式"p"が指すアドレスと式"p+2"が指すアドレスを比較している。
- 式"p"に格納されているアドレスは"0"であるのに対し,"p+2"ではアドレスは"8"を示した。
- よって,p+2 は p の値を 8 バイト増加させた値である。

q10:次の文章は正しいか述べよ。

高水準コード,Q10 ブロック▼

```
88
89      {   //====Q10
90          puts("Q10.次の文章は正しいか述べよ。");
91          puts("    a.1次元配列mは,*mのようにポインタ変数と同じ書式で使用しても良い。");
92          puts("===>同じ書式で使用しても良い。");
93          int m[6] = {0,1,2,3,4,5};
94          printf("    m[0] = %d,     *m = %d\n",m[0],*m);
95          printf("    m[1] = %d, *(m+1) = %d\n\n",m[1],*(m+1));
96          puts("    b.ポインタ変数pは,p[0]のように配列名と同じ書式で使用しても良い。");
97          puts("===>同じ書式で使用しても良い。");
98          int *p;
99          p = m;
100         printf("    p[0] = %d,     *p = %d\n",p[0],*p);
101         printf("    p[1] = %d, *(p+1) = %d\n\n",p[1],*(p+1));
102     }
```

該当箇所の実行結果▼

```
Q10.次の文章は正しいか述べよ。
    a.1次元配列mは,*mのようにポインタ変数と同じ書式で使用しても良い。
===>同じ書式で使用しても良い。
    m[0] = 0,     *m = 0
    m[1] = 1, *(m+1) = 1

    b.ポインタ変数pは,p[0]のように配列名と同じ書式で使用しても良い。
===>同じ書式で使用しても良い。
    p[0] = 0,     *p = 0
    p[1] = 1, *(p+1) = 1
```

考察▼

・a 項では 1 次元配列を,逆参照演算子を用いてポインタ変数のような書式で使用できるかどうかを試行している。
・b 項ではポインタ変数"p"に配列のアドレスを入力し,配列と同様の書式で使用することができるかを試行している。
・出力結果のように,いずれの場合も正常に使用できた。
・よって,文章 a,文章 b は共に正しい。

q11:*m, *(m+3), *m+3, *m++(m+3)の値を示せ。

高水準コード,Q11 ブロック▼

```
103
104     {   //====Q11
105         puts("Q11.*m,*(m+3),*m+3,*m++(m+3)の値を示せ。");
106         static int m[5] = {10, 20, 40, 50, 30};
107         puts("===>*mの値は10, *(m+3)の値は50, *m+3の値は13, *m++(m+3)の値は60である。");
108         printf("    *m = %d, *(m+3) = %d, *m+3 = %d, *m++(m+3) = %d\n\n",*m,*(m+3),*m+3,*m++(m+3));
109     }
```

該当箇所の実行結果▼

```
Q11.*m,*(m+3),*m+3,*m++(m+3)の値を示せ。
===>*mの値は10, *(m+3)の値は50, *m+3の値は13, *m++(m+3)の値は60である。
    *m = 10, *(m+3) = 50, *m+3 = 13, *m++(m+3) = 60
```

考察▼

・配列 m は 1 次元配列であるから,"*m"が示す値は配列の先頭に格納されている値である。
・式"*(m+3)"が示すのは,先頭アドレスを示す m の値を 3 つずらした先に格納されている値,つまり"50"である。
・式"*m+3"が示すのは,"*m"が示している配列の先頭の値に,値"3"を足した値,つまり"13"である。
・式"*m++(m+3)"が示すのは,"*m"が示している配列の先頭の値と,"*(m+3)"が示している値"50"を足した値
つまり"60"である。

q12:*d[2], *(d[2]+2), *d[2]+2, **d, *(*d+3), **d+6, *(d[1]+2), **(d+2)の値を示せ。

高水準コード,Q12 ブロック▼

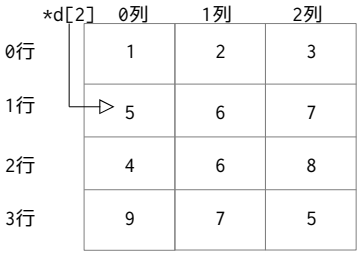
```
110
111     {   //====Q12
112         puts("Q12.*d[2],*(d[2]+2),*d[2]+2,**d,*(*d+3),**d+6,*(d[1]+2),**(d+2)の値を示せ。");
113         static int d[][3] = {{1,2,3},{5,6,7},{4,6,8},{9,7,5}};
114         printf("===>*d[2]の値は4, *(d[2]+2)の値は8, *d[2]+2の値は6, **dの値は1,\n");
115         printf("    *(*d+3)の値は5, **d+6の値は7, *(d[1]+2)の値は7, **(d+2)の値は4である。\n");
116         printf("    *d[2]   = %d, *(d[2]+2) = %d,   *d[2]+2 = %d,     **d = %d,\n    *(*d+3) = %d,     **d+6 = %d,
    *(d[1]+2) = %d, **(d+2) = %d\n\n",*d[2],*(d[2]+2),*d[2]+2,**d,*(*d+3),**d+6,*(d[1]+2),**(d+2));
117     }
```

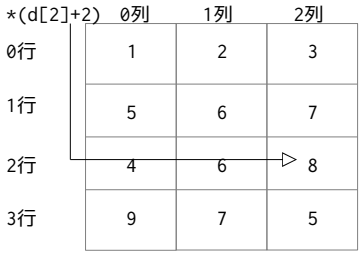
該当箇所の実行結果▼

```
Q12.*d[2],*(d[2]+2),*d[2]+2,**d,*(*d+3),**d+6,*(d[1]+2),**(d+2)の値を示せ。
===>*d[2]の値は4, *(d[2]+2)の値は8, *d[2]+2の値は6, **dの値は1,
    *(*d+3)の値は5, **d+6の値は7, *(d[1]+2)の値は7, **(d+2)の値は4である。
    *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) = 4
```

考察▼

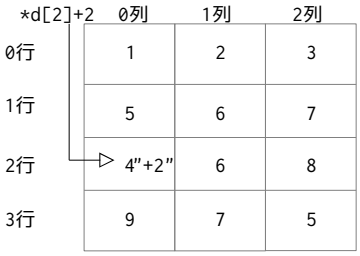
・式"*d[2]"について,0 番目から数えて 2 番目の行の先頭の値を示している。

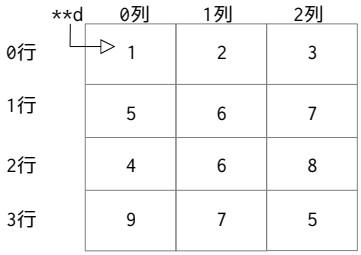




・式"*(d[2]+2)について,"d[2]"は 0 番目から数えて 2 番目の行を示し,その配列の中で"+2"分対象を移した値,つまり"8"を参照している。

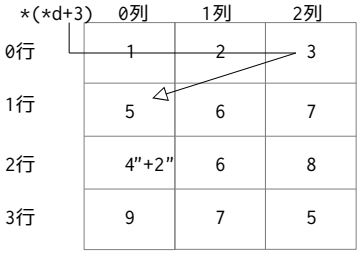
・式"*d[2]+2"について"*d[2]"は 0 番目から数えて 2 番目の行の先頭の値に"+2"を足した値,つまり"6"である。

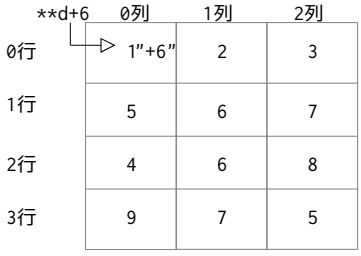




・式"**d"について,"*d"は 2 次元配列のうち最初の 1 次元配列を示し,残る逆参照演算子で示された配列の先頭の値,つまり"1"を示している。

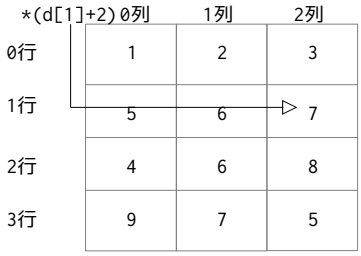
・式"*(*d+3)"について,"*d"は 2 次元配列のうち最初の 1 次元配列を示し,その配列の中で"+3"分対象を移した値を示すが,"+3"分対象を移すと"*d"が示した 1 次元配列の次の配列に対象が移る。よって式"*(*d+3)"は値"5"を取る。

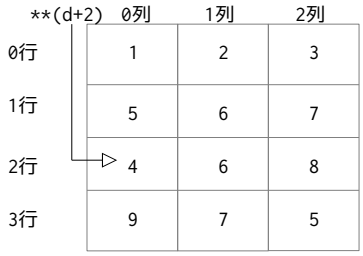




・式"**d+6"について,"**d"は 2 次元配列のうち最初の 1 次元配列の先頭の値を示し,その値に"6"を足した値,つまり"7"を値とする。

・式"*(d[1]+2)"について,"d[1]"は 2 次元配列のうち 0 番目から数えて 1 番目の 1 次元配列の先頭の値を示し,
"+2"分対象を移した値,つまり"7"を示す。





・式"**(d+2)"について,"d"は 2 次元配列のうち 0 番目の 1 次元配列を示し,"+2"分アドレスの対象を移し
逆参照演算子によってそこに格納されている値,すなわち"4"を示している。

q13:ポインタ変数 p の文字列を示せ。

高水準コード,Q13 ブロック▼

```
118
119     {   //====Q13
120         puts("Q13.ポインタ変数pの文字列を示せ。");
121         char *str = "abcdefg", *p;
122         p = str + 3;
123         puts("===>ポインタ変数pの文字列は defg である。");
124         printf("    p = %s\n\n",p);
125     }
```

該当箇所の実行結果▼

```
Q13.ポインタ変数pの文字列を示せ。
===>ポインタ変数pの文字列は defg である。
    p = defg
```

考察▼

・122 行目では,文字列が入力されているポインタ変数"str"の,先頭を示していたアドレスを
"+3"分ずらした"d"のアドレスがポインタ変数"p"に入力されている。
・出力結果は"defg"となっていることから,変換指定子"%s"は示されているアドレスから出力が始まることが分かった。

q14:p,*p,*(p+2)の値を示せ。

高水準コード,Q14 ブロック▼

```
126
127     {   //====Q14
128         puts("Q14.p,*p,*(p+2)の値を示せ。");
129         char *p;
130         p = "abc";
131         puts("===>pの値は文字列の先頭アドレスあるいは文字列を示し, *pはaを, *(p+2)はcを示す。");
132         printf("    p(x) = %x, p(s) = %s, *p = %c, *(p+2) = %c\n\n",p,p,*p,*(p+2));
133     }
```

該当箇所の実行結果▼

```
Q14.p,*p,*(p+2)の値を示せ。
===>pの値は文字列の先頭アドレスあるいは文字列を示し, *pはaを, *(p+2)はcを示す。
    p(x) = df1ba42, p(s) = abc, *p = a, *(p+2) = c
```

考察▼

・文字列の先頭あるいは中間の文字を指すアドレスは,"%s"として出力することで"NULL"が現れるまでのメモリに
格納されている文字を文字列として出力することができた。
・逆参照演算子を付けると,文字列ではなく 1 文字として考えられる。
・式"*(p+2)"は,文字列先頭を示す"p"のアドレスを"+2"分増加させた値,つまり"c"を参照している。

q15:*m,*p,*q の値を示せ。

高水準コード,Q15 ブロック▼

```
134
135     {   //====Q15
136         puts("Q15.*m,*p,*qの値を示せ。");
137         static char m[] = "abcd";
138         char *p,*q;
139         p = &m[0];
140         q = m;
141         puts("===>*mの値は文字aを, *pの値は文字列abcdの先頭アドレスを,*qの値は文字列の先頭アドレスを示す。");
142         printf("    *m = %c, *p = %c, *q = %c\n\n",*m,*p,*q);
143     }
```

該当箇所の実行結果▼

```
Q15.*m,*p,*qの値を示せ。
===>*mの値は文字aを, *pの値は文字列abcdの先頭アドレスを,*qの値は文字列の先頭アドレスを示す。
    *m = a, *p = a, *q = a
```

考察▼

・文字型配列 m は 1 次元配列であるから,式"m"は配列の先頭アドレス及び文字列を示す。
・式"*m"は,逆参照演算子によって先頭アドレスではなく配列に格納されている文字 1 文字ずつ,この場合は"a"を示す。
・ポインタ変数"p"には"&m[0]",つまり 1 次元配列 m の 0 番目の値へのアドレスが入力されている。
・よってその値を参照すると,配列 m の 0 番目の値,"a"が得られる。
・ポインタ変数"q"には,"m"つまり配列の先頭アドレスが入力されているが,1 文字としてしか認識されない。
・文字列として変換指定子"%s"を使って出力しようとすると"segmentation fault"が現れる。

q16:*p,*(m+2),*m+2 の値を示せ。

高水準コード,Q16 ブロック▼

```
144
145     {   //====Q16
146         puts("Q16.*p,*(m+2),*m+2の値を示せ。");
147         static char m[] = "abcd";
148         char *p;
149         p = &m[2];
150         puts("===>*pの値はc, *(m+2)の値はc, *m+2の値はcである。");
151         printf("    *p = %c, *(m+2) = %c, *m+2 = %c\n\n",*p,*(m+2),*m+2);
152     }
```

該当箇所の実行結果▼

```
Q16.*p,*(m+2),*m+2の値を示せ。
===>*pの値はc, *(m+2)の値はc, *m+2の値はcである。
    *p = c, *(m+2) = c, *m+2 = c
```

考察▼

・ポインタ変数"p"には配列 m の 0 番目から数えて 2 番目のアドレスが入力されているため,ポインタ変数"p"の値を参照すると文字"c"が出力される。
・式"*(m+2)"について,"m"は配列 m の先頭あるいは文字列を表し,"+2"分表す対象をずらした値は文字"c"である。
・式"*m+2"について,"*m"は配列 m の先頭文字を表しているため"a"を指しているが,"+2"で文字コードとして値が増加しているため最終的に表す値は"c"になっている。

q17:ポインタ変数 p の文字列を示せ。

高水準コード,Q17 ブロック▼

```
153
154     {   //====Q17
155         puts("Q17.ポインタ変数pの文字列を示せ。");
156         char m[]={"abcd"};
157         char *p;
158         p = m;
159         *(p+1) = 'x';
160         puts("===>ポインタ変数pの文字列は axcdである。");
161         printf("    p(s) = %s\n\n",p);
162     }
```

該当箇所の実行結果▼

```
Q17.ポインタ変数pの文字列を示せ。
===>ポインタ変数pの文字列は axcdである。
    p(s) = axcd
```

考察▼

・問題文そのままでは"segmentation fault"が発生するため,新たに文字型配列 m[]を宣言した。
・ポインタ変数に配列のアドレスを入力することで,159 行目のようなポインタ的表記で文字列を変更することができる。

q18:変数 x の値を示せ。

高水準コード,Q18 ブロック▼

```
163
164     {   //====Q18
165         puts("Q18.変数xの値を示せ。");
166         int x;
167         char *p;
168         p = "abcd";
169         if ( p == "abcd") x = 0;
170         else              x = 1;
171         puts("===>変数xの値は0である。");
172         printf("    x = %d\n\n",x);
173     }
```

該当箇所の実行結果▼

```
Q18.変数xの値を示せ。
===>変数xの値は0である。
    x = 0
```

考察▼

・文字列型ポインタ変数"p"に入力された値は,文字列として if 文の条件に使うことができる。

q19:ポインタの代わりに"int k"を宣言し,配列を用いた文に書き換えよ。

書き換え前のコード▼

```
char m[MAX], *p;
for(p=m; *p; ++p) *p += 1;
```

高水準コード,Q19 ブロック▼

```
174
175     {   //====Q19
176         #define MAX 256
177         puts("Q19.ポインタの代わりに\"int k\"を宣言し,配列を用いた文に書き換えよ。");
178         char m[25];
179         int k;
180         for ( k=0 ; m[k] == 0 ; k++){
181             m[k]+=1;
182         }
183         printf("====k = %d\n\n",k);
184     }
```

該当箇所の実行結果▼

```
Q19.ポインタの代わりに"int k"を宣言し,配列を用いた文に書き換えよ。
====k = 256
```

考察▼

・書き換え前のコードはポインタ変数"p"に 1 次元配列"m"の先頭アドレスを入力している。
・"*p"に 0 以外の値は入っていることを条件にループされ,繰り返す内容は"*p"に 1 を足す処理である。
・更に処理終了後に"p"がインクリメントされている。
・書き換え後のコードでは,題意通り新たに"int k"を宣言して処理を再現した。

q20:*q[2],q[3][2],*(q[2]+2),*(*(q+3)+2),**(q+1)の値を示せ。

高水準コード,Q20 ブロックからコード最後まで▼

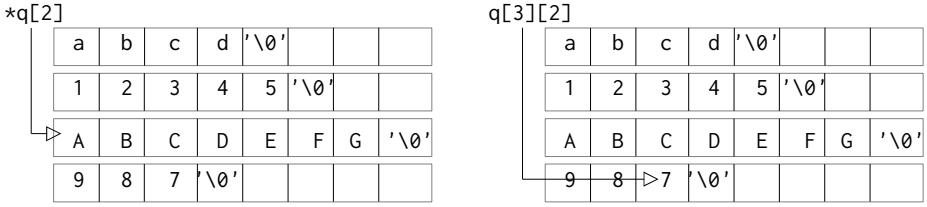
```
185
186     {   //====Q20
187         puts("Q20.*q[2],q[3][2],*(q[2]+2),*(*(q+3)+2),**(q+1)の値を示せ。");
188         static char *q[] = {"abcd", "12345", "ABCDEFG", "987"};
189         puts("===>*q[2]の値は文字\"A\"を, q[3][2]は文字\"7\"を, *(q[2]+2)は文字\"C\"を,\n    *(*(q+3)+2)は文字と
190 ての\"7\"を, **(q+1)は文字としての\"1\"を示す。");
191         printf("    *q[2] = %c, q[3][2] = %c, *(q[2]+2) = %c, *(*(q+3)+2) = %c, **(q+1) = %c\n\n",*q[2],q[3]
192 [2],*(q[2]+2),*(*(q+3)+2),**(q+1));
193     }
194     return(0);
195
196 }
```

該当箇所の実行結果▼

```
Q20.*q[2],q[3][2],*(q[2]+2),*(*(q+3)+2),**(q+1)の値を示せ。
===>*q[2]の値は文字"A"を, q[3][2]は文字"7"を, *(q[2]+2)は文字"C"を,
    *(*(q+3)+2)は文字としての"7"を, **(q+1)は文字としての"1"を示す。
    *q[2] = A, q[3][2] = 7, *(q[2]+2) = C, *(*(q+3)+2) = 7, **(q+1) = 1
```

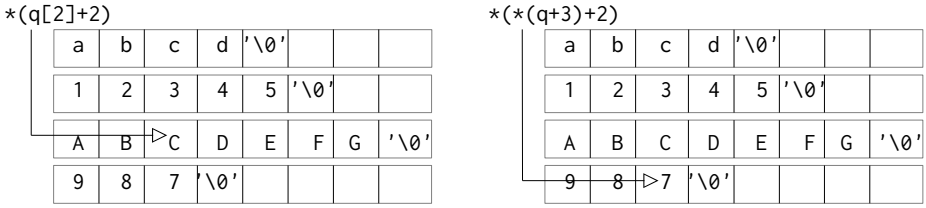
考察▼

・式"*q[2]"について,2 次元配列 q の 0 番目から数えて 2 番目の 1 次元配列を示し,その先頭の値である"A"を参照している。

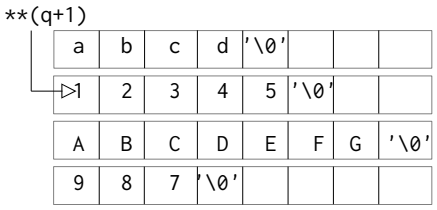


・式"q[3][2]"について,2 次元配列 q の 0 番目から数えて 3 番目の 1 次元配列を示した後,その配列の中で 0 番目から数えて 2 番目にある文字"7"を示している。

・式"*(q[2]+2)"について,2 次元配列 q の 0 番目から数えて 2 番目の 1 次元配列を示した後,"+2"分示す対象をずらした値が"C"である。



・式"*(*(q+3)+2)"について,2 次元配列 q の 0 番目から数えて 3 番目の 1 次元配列の先頭アドレスを示した後,"+2"分示す対象をずらした値が"7"である。
・式"**(q+1)"について,2 次元配列の 0 番目から数えて 1 番目の 1 次元配列を示した後,
1 度目の逆参照演算子でその先頭アドレスおよび配列を,2 度目の逆参照演算子で 1 文字"1"を表している。



2.リスト構造について考察せよ。

高水準コードの一部▼

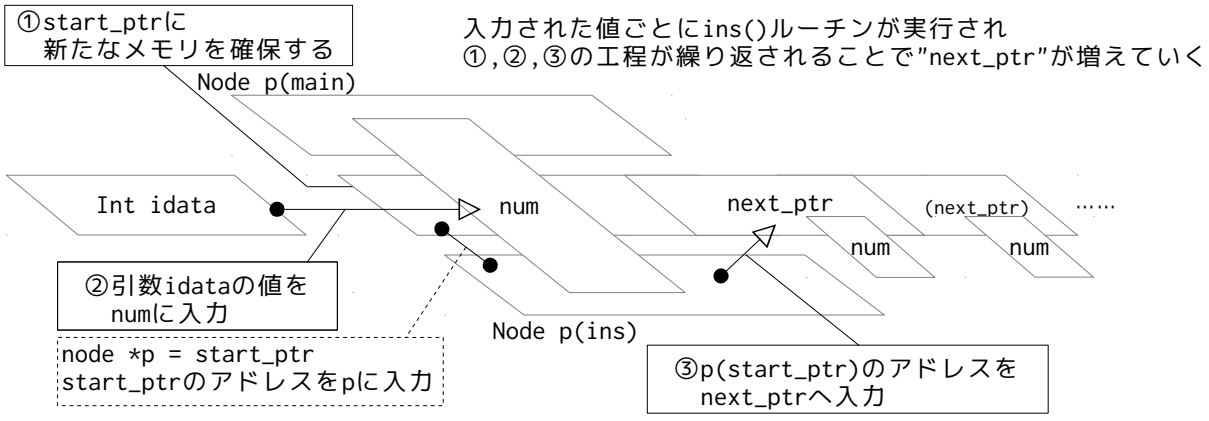
```
8   #include <stdio.h>
9   #include <stdlib.h>
10
11  #define FALSE 0                     // FALSEを0に置換
12  #define TRUE  !FALSE                // TRUEを!FALSE(!0)に置換
13
14  typedef struct Node{                // typedefによって構造体struct Nodeを宣言
15      int num;                        // 構造体内でint型変数numを宣言
16      struct Node *next_ptr;          // 構造体自身を示すポインタ*next_ptrを宣言
17  }node;
18
19  node *start_ptr = NULL;             // 自己参照型構造体として*start_ptrを宣言し,NULLで初期化する
20
21  void ins(int idata){                // サブルーチンinsにint型変数 idataを引き渡して実行
22      node *p = start_ptr;            // 自己参照型構造体として*pを宣言し,start_ptrに格納されているアドレスを入力
23
24      start_ptr = (node *)malloc(sizeof(node));
25                                      // 構造体start_ptrに,node分のサイズのメモリを確保しそのアドレスを入力
26      if (start_ptr == NULL) puts("Not enough memory!"), exit(0);
27                                      // 構造体のメモリが確保されていなければプログラムを終了
28
29      start_ptr->num = idata;         // 構造体start_ptrの中のint変数numに idataの値を入力
30      start_ptr->next_ptr = p;        // 構造体start_ptrの中の構造体next_ptrに pのアドレスを渡す
31  }
32
33  int main(){
34      int idata;                      // int型変数idataを宣言
35      node *p;                        // 自己参照型構造体*pを宣言
36
37      puts("Enter a sequence of integers:");          // メッセージを表示
38      while(scanf("%d", &idata) == TRUE) ins(idata);
39                                      // 基数10でidataのアドレスが示す場所に値を取り込みその値が0以外であれば
40                                      // サブルーチンins()にidataを引数として引き渡す
41      puts("In reverse order:");      // メッセージを表示
42      for(p = start_ptr; p != NULL; p = p->next_ptr){ //ポインタpに構造体start_ptrのアドレスを入力し
43                                      //pの値がNULLでなければ繰り返し, pに構造体pの中のnext_ptrの値を入力
44          printf("%5d-", p->num);     /* アドレスと値の出力へ更新しなさい */
45      }
46      puts("/end/");                  //メッセージを表示
47
47      return(0);                      //プログラムを終了
48  }
```

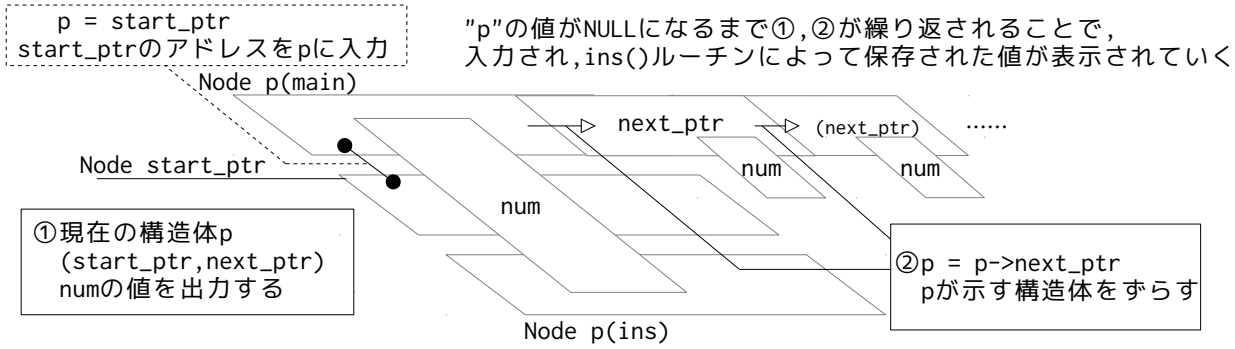
該当箇所の実行結果▼ ※整数 293,443,38,221,12,34 を入力し,直後に Control+D を入力している。

```
Enter a sequence of integers:
293
443
38
221
12
34
(^D)
In reverse order:
   34-   12-  221-   38-  443-  293-/end/
```

考察▼

・14 行目から始まっている typedef による構造体の宣言のような表記は,ユーザーで変数（構造体）の"型"を宣言している。
・宣言することで,それ以降"node"という型が有効になり,構造体を宣言するときであっても"node［構造体名］"とするだけでよい
・28 行目において,現在の構造体のアドレスを"次の構造体"のアドレスに入力している。
・これが入力した値が逆に入力される仕組みであり,構造体"start_ptr"と ins()ルーチン内で宣言されたポインタ構造体"p"に着目すると以下のような流れが行われている。





XXX.あとがき（反省・感想・参考）。

a-1.参考サイト・文献。

- 『C 実践プログラミング 第三版』（オーム社）
- 『Open Lecture』
  http://www.osn.u-ryukyu.ac.jp/lecture/wiki/index.php?Open%20Lecture
- 『C Programming』
  http:://www.kusa.ac.jp/~kajiura/c/main/newpage32.htm

a-2.反省・感想。

問題・解答を確かめながら,配列・ポインタの関わりあいを確認することができました。
ごちゃごちゃになりがちなポインタをしっかり理解して,次のレポートに挑みたいです。

今回のリスト構造は,次回のソートの前哨戦といった感じがしました。
ポインタに更に構造体が絡んでいて,言葉だけでは理解しきれない,の意味が分かりました…。
最後までご覧頂きありがとうございました。