プログラミング 1

Report#7

提出日　：2013 年 7 月 11 日
所　属　：工学部情報工学科
学籍番号：e135732J
氏　名　：前城　健太郎

Report#7

資料「問題と解答」の「II ポインタとアドレス」より、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。併せて、リスト構造についても考察せよ。

１．解答確認プログラムの制作と考察

1.1 (Q1)変数ｖのアドレスを求める式を示せ。

1.1.1 プログラムの先頭と Q01 ブロックのソースコード

```
#include <stdio.h>
int main(){

  /* Q01 */
  {
    puts("Q01.変数ｖのアドレスを求める式を示せ");
    puts("A01.変数ｖのアドレスを求める式は&v である");
    int v = 4;
    int *p;
    p = &v;
    printf("&v ==> %08x\n",&v);
    printf("*p ==> %d\n",*p);
  }

```

1.1.2 該当箇所の実行結果

```
Q01.変数ｖのアドレスを求める式を示せ
A01.変数ｖのアドレスを求める式は&v である
&v ==> 517729bc
*p ==> 4
```

1.1.3 考察

a) 変数ｖのアドレスを求める式は&v である.

b) 実際にポインタ変数ｐに変数ｖのアドレスを格納し*p を表示すると,変数ｖに格納されている数が表示された.

1.2 (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。

1.2.1 Q02 ブロックのソースコード

```
15    /* Q02 */
16    {
17      puts("Q02.1 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。");
18      puts("A02.1 つ目の式は&m[5],2 つ目の式は m+5 である");
19      int m[5];
20      printf("&m[5] ==> %08x\n",&m[5]);
21      printf("m+5   ==> %08x\n",m+5);
22    }
23
```

1.2.2 該当箇所の実行結果

```
Q02.1 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。
A02.1 つ目の式は&m[5],2 つ目の式は m+5 である
&m[5] ==> 517729b0
m+5   ==> 517729b0
```

1.2.3 考察

a) アドレスを求めるアドレスはそれぞれ&m[5],m+5 である.

b) 実行結果より,それぞれの結果は同じであることがわかる.

1.3 (Q3)1 次元配列 m の先頭アドレスを求める式を、2 つ示せ。

1.3.1 Q03 ブロックのソースコード

```
24    /* Q03 */
25    {
26      puts("Q03.1 次元配列 m の先頭アドレスを求める式を、2 つ示せ。");
27      puts("A03.1 つ目の式は&m[0],2 つ目の式は m である");
28      int m[5];
29      printf("&m[0] ==> %08x\n",&m[0]);
30      printf("m     ==> %08x\n",m);
31    }
32
```

1.3.2 該当箇所の実行結果

```
Q03.1 次元配列 m の先頭アドレスを求める式を、2 つ示せ。
A03.1 つ目の式は&m[0],2 つ目の式は m である
&m[0] ==> 51772988
m     ==> 51772988
```

1.3.3 考察

a) アドレスを求めるアドレスはそれぞれ&m[0],m である.

b) 実行結果より,それぞれの結果は同じであることがわかる.

1.4 (Q4)2 次元配列 d の先頭アドレスを求める式を、3 つ示せ。

1.4.1 Q04 ブロックのソースコード

```
33    /* Q04 */
34    {
35      puts("Q04.2 次元配列 d の先頭アドレスを求める式を、3 つ示せ。");
36      puts("A04.1 つ目の式は&d[0][0],2 つ目の式は d[0],3 つ目の式は*d である");
37      int d[3][4];
38      printf("&d[0[0] ==> %08x\n",&d[0][0]);
39      printf("d[0]    ==> %08x\n",d[0]);
40      printf("*d      ==> %08x\n",*d);
41      printf("d       ==> %08x\n",d);
42    }
43
```

1.4.2 該当箇所の実行結果

```
Q04.2 次元配列 d の先頭アドレスを求める式を、3 つ示せ。
A04.1 つ目の式は&d[0][0],2 つ目の式は d[0],3 つ目の式は*d である
&d[0[0] ==> 5e450958
d[0]    ==> 5e450958
*d      ==> 5e450958
d       ==> 5e450958
```

1.4.3 考察

a) アドレスを求める式はそれぞれ&d[0][0],d[0],*d である.ただし,d であっても先頭のアドレスを表す.
b) 2 次元配列の場合,&d[0][0]は「配列 d の 0 行 0 列目の要素のアドレス」,d[0]は「配列 d の 0 列目の先頭要素のアドレス」,d は「配列 d の先頭要素のアドレス」とみなすことができる.
c) また,*d というポインタが示す対象は配列でありポインタではないから,間接参照が生成されず d と同じような結果となる.

1.5 (Q5)次の文を実行した後の変数 a の値を示せ。

1.5.1 Q05 ブロックのソースコード

```
44    /* Q05 */
45    {
46      puts("Q05.実行した後の変数 a の値を示せ。");
47      puts("A05.変数 a の値は 8 である");
48      int a=2, b=3, c=5, *p, *q;
49      p = &b;
50      q = &c;
51      a = *p + *q;
52      printf("a ==> %d\n",a);
53    }
54
```

1.5.2 該当箇所の実行結果

```
Q05.実行した後の変数 a の値を示せ。
A05.変数 a の値は 8 である
a ==> 8
```

1.5.3 考察

a) 49 行目,50 行目において変数 b,c のアドレスがポインタ変数に格納されている.
b) なので,*p,*q の値はそれぞれ変数 b,c を示しているので 3 と 5 である.
c) よって,変数 a には 3 と 5 を足した値 8 が格納される.

1.6 (Q6)次の文を実行した後の変数 a の値を示せ。

1.6.1 Q06 ブロックのソースコード

```
55    /* Q06 */
56    {
57    puts("Q06.実行した後の変数 a の値を示せ。");
58    puts("A06.変数 a の値は 5 である");
59    int a=2, *p;
60    p = &a;
61    *p = 5;
62    printf("a ==> %d\n",a);
63    }
64
```

1.6.2 該当箇所の実行結果

```
Q06.実行した後の変数 a の値を示せ。
A06.変数 a の値は 5 である
a ==> 5
```

1.6.3 考察

a) 60 行目においてポインタ変数 p に変数 a の値を格納している.
b) 61 行目においては,ポインタ変数 p が指すもの,つまり変数 a に数値 5 を格納している.
c) よって,変数 a の値は 5 である.

1.7 (Q7)実行した後の*p、*q、**q の値を示せ。

1.7.1 Q07 ブロックのソースコード

```
65    /* Q07 */
66    {
67      puts("Q07.実行した後の*p,*q,**q の値を示せ。");
68      puts("A07.*p は 200,*q は 200,**q は 300 である");
69      int *p, **q, *q1;
70      int a = 200, b = 300;
71      p  = &a;
72      q1 = &b;
73      q  = &q1;
74      printf("アドレス&a をアドレス 100,アドレス&b をアドレス 200 と考える\n");
75      printf("&a  ==> %08x\n",&a);
76      printf("&b  ==> %08x\n",&b);
77      puts("-----");
78      printf("*p  ==> %d\n",*p);
79      printf("*q  ==> %08x\n",*q);
80      printf("**q ==> %d\n",**q);
81    }
82
```

1.7.2 該当箇所の実行結果

```
Q07.実行した後の*p,*q,**q の値を示せ。
A07.*p は 200,*q は 200,**q は 300 である
アドレス&a をアドレス 100,アドレス&b をアドレス 200 と考える
&a  ==>  5e45090c
&b  ==>  5e450908
-----
*p  ==> 200
*q  ==> 5e450908
**q ==> 300
```

1.7.3 考察

a) 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 q1 に入力した.

b) 問題にない新たなポインタ q1 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.

c) p に変数 a のアドレス,q に変数 b のアドレスを格納することで,問題文の条件と同じ状況を再現している.

d) *p は変数 a を指しているので値は 200,q は 2 重ポインタであり,*q は q1 のアドレス,つまり変数 b のアドレスを指しているので 200.

e) **q は変数 b を指しているので 300 となる.

1.8 (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。

1.8.1 Q08 ブロックのソースコード

```
83    /* Q08 */
84    {
85      puts("Q08.1 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。");
86      puts("A08.m[k]は k 番目の配列の値を示し、*(m+k)も同じく k 番目の値を示す");
87      int m[10]={0,1,2,3,4,5,6,7,8,9};
88      int k = 6;
89      printf("m[k]   ==> %d\n",m[k]);
90      printf("*(m+k) ==> %d\n",*(m+k));
91    }
92
```

1.8.2 該当箇所の実行結果

```
Q08.1 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。
A08.m[k]は k 番目の配列の値を示し、*(m+k)も同じく k 番目の値を示す
m[k]   ==> 6
*(m+k) ==> 6
```

1.8.3 考察

a) 1 次元配列の場合,m[k]は k 番目の要素を表す.
b) また,m は 1 次元配列 m の先頭アドレスを表し,m+k によって k 番目のアドレスとなる.ここでは,増加するアドレスの量は自動的に決まる.
c) よって,*(m+k)は m+k が指す値,つまり k 番目の要素を指している.

1.9 (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。

1.9.1 Q09 ブロックのソースコード

```
93      /* Q09 */
94      {
95        puts("Q09.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。");
96        puts("A09.整数型ポインタ変数の場合、8 バイト増加させた値である");
97        int *p;
98        printf("p   ==> %08x\n",p);
99        printf("p+2 ==> %08x\n",p+2);
100       char *q;
101       puts("char 型の場合");
102       printf("q   ==> %08x\n",q);
103       printf("q+2 ==> %08x\n",q+2);
104     }
105
```

1.9.2 該当箇所の実行結果

```
Q09.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。
A09.整数型ポインタ変数の場合、8 バイト増加させた値である
p   ==> 00000000
p+2 ==> 00000008
char 型の場合
q   ==> 00000000
q+2 ==> 00000002
```

1.9.3 考察

a) 整数型ポインタ変数の場合,確保されるアドレスのサイズは 4 バイトであるので p+2 は 8 バイト増加する.

b) ちなみに,char 型の場合は確保されるアドレスは 1 バイトであるので,q+2 は 2 バイト増加する.

1.10 (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。
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。

1.10.1 Q10 ブロックのソースコード

```
106    /* Q10 */
107    {
108      puts("Q10.次の文章は正しいか述べよ。\na.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。\nb.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。");
109      puts("A10.2 つとも正しい");
110      int m[6] = {0,1,2,3,4,5};
111      puts("a.");
112      printf(" m[0] ==> %d, *m ==> %d\n",m[0],*m);
113      printf(" m[1] ==> %d, *(m+1) ==> %d\n",m[1],*(m+1));
114      int *p;
115      p = m;
116      puts("b.");
117      printf(" p[0] ==> %d, *p ==> %d\n",p[0],*p);
118      printf(" p[1] ==> %d, *(p+1) ==> %d\n",p[1],*(p+1));
119    }
120
```

1.10.2 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。
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。
A10.2 つとも正しい
a.
 m[0] ==> 0, *m ==> 0
 m[1] ==> 1, *(m+1) ==> 1
b.
 p[0] ==> 0, *p ==> 0
 p[1] ==> 1, *(p+1) ==> 1
```

1.10.3 考察

a) 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) 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.
c) 出力結果のように,いずれの場合も正常に使用できた.

1.11 (Q11)次の定義による、*m、*(m+3)、*m+3、*m+*(m+3) の値を示せ。

1.11.1 Q11 ブロックのソースコード

```
121     /* Q11 */
122     {
123       puts("Q11.次の定義による値を示せ");
124       puts("A11.それぞれ*m=10、*(m+3)=50、*m+3=13、*m+*(m+3)=60");
125       static int m[5] = {10, 20, 40, 50, 30};
126       printf("*m        ==> %2d\n",*m);
127       printf("*(m+3)    ==> %2d\n",*(m+3));
128       printf("*m+3      ==> %2d\n",*m+3);
129       printf("*m+*(m+3) ==> %2d\n",*m+*(m+3));
130     }
131
```

1.11.2 該当箇所の実行結果

```
Q11.次の定義による値を示せ
A11.それぞれ*m=10、*(m+3)=50、*m+3=13、*m+*(m+3)=60
*m        ==> 10
*(m+3)    ==> 50
*m+3      ==> 13
*m+*(m+3) ==> 60
```

1.11.3 考察

a) *m について.配列 m の先頭の要素を指しているので値は 10 である.
b) *(m+3)について.1.8 項の実行結果からわかるように,この場合先頭アドレス m に 3 を足しているので 3 番目の要素を指している,よって,値は 50 となる.
c) *m+3 について.*(m+3)とは違いカッコが無いので*m から先に処理される.*m の値は 10 であるので,*m+3 の値は 13 となる.
d) *m+*(m+3)について.*m と*(m+3)から先に処理され,次にそれぞれの値を足している.よって,値は 60 である.

1.12 (Q12)次の定義による、
d[2]、(d[2]+2)、*d[2]+2, **d、*(*d+3)、**d+6、*(d[1]+2)、**(d+2)
の値を示せ。

1.12.1 Q12 ブロックのソースコード

```
132    /* Q12 */
133    {
134      puts("Q12.次の定義による値を示せ");
135      puts("A12.それぞれ*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");
136      static int d[][3] = {{1,2,3}, {5,6,7}, {4,6,8}, {9,7,5}};
137      printf("*d[2]     ==> %d\n",*d[2]);
138      printf("*(d[2]+2) ==> %d\n",*(d[2]+2));
139      printf("*d[2]+2   ==> %d\n",*d[2]+2);
140      printf("**d       ==> %d\n",**d);
141      printf("*(*d+3)   ==> %d\n",*(*d+3));
142      printf("**d+6     ==> %d\n",**d+6);
143      printf("*(d[1]+2) ==> %d\n",*(d[1]+2));
144      printf("**(d+2)   ==> %d\n",**(d+2));
145    }
146
```

1.12.2 該当箇所の実行結果

```
Q12.次の定義による値を示せ
A12.それぞれ*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
*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
```

1.12.3 考察

a) *d[2]について.d[2]の先頭の要素を指しているので値は 4 である.
b) *(d[2]+2)について.d[2]の先頭アドレスに 2 を足してる,つまり d[2]の 2 番目の要素を指しているので値は 8 である.これは,d[2][2]と同じ値である.
c) *d[2]+2 について.*d[2]の値自体に 2 を足してるので,値は 6 である.
d) **d について.配列 d の先頭要素を指してるので,値は 1 である.これは,d[0][0]と同じ値である.
e) *(*d+3)について.*d は配列 d の 0 行目先頭アドレスを指し,これに 3 足している.2 次元配列の場合後ろの数からメモリが割り当てられてるので,この場合,
d[0][0]→d[0][1]→d[0][2]→d[1][0]というように添字は増加する.よって,値は 5 である.考察は後述する.
f) **d+6 について.配列 d の先頭要素に 6 を足しているので,値は 7 である.
g) *(d[1]+2)について.d[1]は 1 行目先頭アドレスを表しそれに 2 を足している.よってこれは d[1][2]と同じ値を表し,その値は 7 である.
h) **(d+2)について.配列 d の先頭アドレスに 2 を足している.これは d[2][0]の要素のアドレスと同じである.よって値は 4 である.考察は後述する.

1.12.4 ２次元配列のメモリの観察

1.12.4.1 ソースコード

```
1   #include <stdio.h>
2
3   int main(){
4
5     int d[4][3] = {{1,2,3},{5,6,7},{4,6,8},{9,7,5}};
6
7     printf("d ==> %08x\n",d);
8
9     printf("&d[0][0] ==> %08x\n",&d[0][0]);
10    printf("&d[0][1] ==> %08x\n",&d[0][1]);
11    printf("&d[0][2] ==> %08x\n",&d[0][2]);
12    printf("&d[1][0] ==> %08x\n",&d[1][0]);
13    printf("&d[2][0] ==> %08x\n",&d[2][0]);
14    printf("&d[3][0] ==> %08x\n",&d[3][0]);
15    puts("-----");
16    printf("*(d+1)   ==> %08x\n",*(d+1));
17    printf("*(d+2)   ==> %08x\n",*(d+2));
18    printf("*(d+3)   ==> %08x\n",*(d+3));
19    puts("-----");
20    printf("(*d+1)   ==> %08x\n",(*d+1));
21    printf("(*d+2)   ==> %08x\n",(*d+2));
22    printf("(*d+3)   ==> %08x\n",(*d+3));
23
24    return(0);
25  }
```

1.12.4.2 実行結果

```
d ==> 5ccbfa18
&d[0][0] ==> 5ccbfa18
&d[0][1] ==> 5ccbfa1c
&d[0][2] ==> 5ccbfa20
&d[1][0] ==> 5ccbfa24
&d[2][0] ==> 5ccbfa30
&d[3][0] ==> 5ccbfa3c
-----
*(d+1)   ==> 5ccbfa24
*(d+2)   ==> 5ccbfa30
*(d+3)   ==> 5ccbfa3c
-----
(*d+1)   ==> 5ccbfa1c
(*d+2)   ==> 5ccbfa20
(*d+3)   ==> 5ccbfa24
```

1.12.4.3 考察

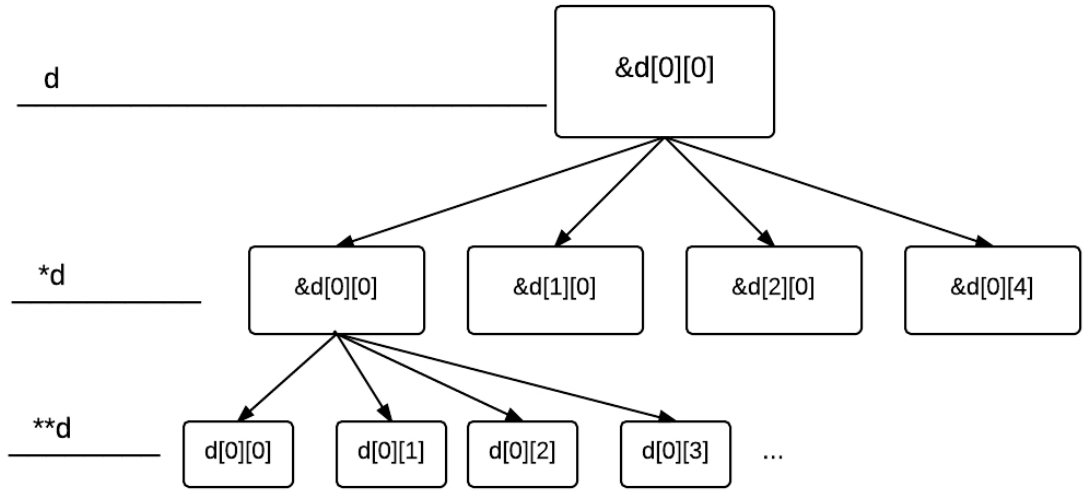
a) 2 次元配列 d の 0 列目,1 列目,2 列目と 0 行目,1 行目,2 行目の先頭アドレスを表示させた.
b) アドレス d を増加させた場合,行の先頭アドレスを増加させた場合と同じように増加した.
c) また,アドレス*d を増加させた場合,列の先頭アドレスを増加させた場合と同じように増加した.
d) よって,アドレス d は行の先頭アドレスを指し,アドレス*d は列の先頭アドレスを指していることがわかる.
e) また,先頭アドレス d の次のアドレスは d[0][1],次は d[0][2]であることがわかる.よって,添字は後ろの方からしていく.
f) さらに,2 次元配列は 2 重ポインタであると解釈することもできる.

1.12.4.4 1.12.1 項の 2 次元配列の図的解釈

int d[4][3]

		↓*d	↓*d+1	↓*d+2
		0 列	1 列	2 列
d→	0 行(d[0])	1	2	3
d+1→	1 行(d[1])	5	6	7
d+2→	2 行(d[2])	4	6	8
d+3→	3 行(d[3])	9	7	5

2 次元配列を 2 重ポインタとして解釈する.



1.13 (Q13)実行した後のポインタ変数 p の文字列を示せ。

1.13.1 Q13 ブロックのソースコード

```
147   /* Q13 */
148   {
149     puts("Q13.実行した後のポインタ変数 p の文字列を示せ。");
150     puts("A13.ポインタ変数 p の文字列は defg である");
151     char *str = "abcdefg", *p;
152     p = str + 3;
153     printf("p ==> %s\n",p);
154   }
155
```

1.13.2 該当箇所の実行結果

```
Q13.実行した後のポインタ変数 p の文字列を示せ。
A13.ポインタ変数 p の文字列は defg である
p ==> defg
```

1.13.3 考察

a) ポインタ変数には p には文字列が格納されてる変数の先頭アドレスに 3 を足したアドレスが格納されている.

b) よって,*p は defg を表す.

1.14 (Q14)実行した後の p、*p、*(p+2)の値を示せ。

1.14.1 Q14 ブロックのソースコード

```
156    /* Q14 */
157    {
158      puts("Q14.実行した後の p、*p、*(p+2)の値を示せ。");
159      puts("A14.それぞれの値は p=100,*p=a,*(p+2)=c である");
160      char *p;
161      p = "abc";
162      printf("&(*p) = %08x : アドレス 100 とする\n",&(*p));
163      printf("p      ==> %08x\n",p);
164      printf("*p     ==> %c\n",*p);
165      printf("*(p+2) ==> %c\n",*(p+2));
166    }
167
```

1.14.2 該当箇所の実行結果

```
Q14.実行した後の p、*p、*(p+2)の値を示せ。
A14.それぞれの値は p=100,*p=a,*(p+2)=c である
&(*p) = 0916bb3e : アドレス 100 とする
p      ==> 0916bb3e
*p     ==> a
*(p+2) ==> c
```

1.14.3 考察

a) p について.p はポインタ変数であり,条件より&(*p)=100 であるので p の値は 100 である.
b) *p について.p の値は操作されていないので,*p は文字列の先頭を表す.よって,値は a である.
c) *(p+2)について.p に 2 を足してるので文字列の 2 番目の文字を表す.よって,値は c である.

1.15 (Q15)実行した後の *m、*p、*q の値を示せ。

1.15.1 Q15 ブロックのソースコード

```
168    /* Q15 */
169    {
170      puts("Q15.実行した後の *m、*p、*q の値を示せ。");
171      puts("A15.それぞれの値は*m=a,*p=a,*q=a である");
172      static char m[] = "abcd";
173      char *p, *q;
174      p = &m[0];
175      q = m;
176      printf("*m ==> %c\n",*m);
177      printf("*p ==> %c\n",*p);
178      printf("*q ==> %c\n",*q);
179    }
180
```

1.15.2 該当箇所の実行結果

```
Q15.実行した後の *m、*p、*q の値を示せ。
A15.それぞれの値は*m=a,*p=a,*q=a である
*m ==> a
*p ==> a
*q ==> a
```

1.15.3 考察

a) *m について.配列 m の先頭要素を指しているので,値は a である.
b) *p について.p には配列 m の先頭要素 m[0]のアドレスが格納されているので,値は a である.
c) *q について.m 自体では先頭要素のアドレスを指しているので q にも配列 m の先頭要素 m[0]のアドレスが格納されてる.よって,値は a である.

1.16 (Q16)実行した後の *p、*(m+2)、*m+2 の値を示せ。

1.16.1 Q16 ブロックのソースコード

```
181    /* Q16 */
182    {
183      puts("Q16.実行した後の *p、*(m+2)、*m+2 の値を示せ。");
184      puts("A16.それぞれの値は*p=c,*(m+2)=c,*m+2=c である");
185      static char m[] = "abcd";
186      char *p;
187      p = &m[2];
188      printf("*p     ==> %c\n",*p);
189      printf("*(m+2) ==> %c\n",*(m+2));
190      printf("*m+2   ==> %c\n",*m+2);
191    }
192
```

1.16.2 該当箇所の実行結果

```
Q16.実行した後の *p、*(m+2)、*m+2 の値を示せ。
A16.それぞれの値は*p=c,*(m+2)=c,*m+2=c である
*p     ==> c
*(m+2) ==> c
*m+2   ==> c
```

1.16.3 考察

a) *p について.p には配列 m の 2 番目の要素のアドレスが格納されているので,値は c である.

b) *(m+2)について.m+2 は配列 m の先頭アドレスに 2 を足しているので配列の 2 番目の要素のアドレスを表している.よって,値は c である.

c) *m+2 について.*m は配列 m の先頭要素 a を表しているが,それに 2 を足した値は ASCII コード表より c を表す.よって,値は c である.

1.17 (Q17)実行した後のポインタ変数 p の文字列を示せ。

1.17.1 Q17 ブロックのソースコード

```
193   /* Q17 */
194   {
195     puts("Q17.実行した後のポインタ変数 p の文字列を示せ。");
196     puts("A17.ポインタ変数 p の文字列は\"axcd\"である");
197     char m[] = {"abcd"};        /* bus error を防ぐため */
198     char *p;
199     p = m;
200     *(p + 1) = 'x';
201     printf("p ==> %s\n",p);
202   }
203
```

1.17.2 該当箇所の実行結果

```
Q17.実行した後のポインタ変数 p の文字列を示せ。
A17.ポインタ変数 p の文字列は"axcd"である
p ==> axcd
```

1.17.3 考察

a) 問題の文をそのまま実行すると bus error を起こすので,197 行目において文字列 abcd のメモリを確保している.
b) *(p+1)について.これは配列 m の 1 番目の要素を表している.
c) よって,そこへ x を格納することで 1 番目の要素だけを変更することになる.
d) なので,答えの値は axcd である.

1.18 (Q18)次の文を実行した後の変数 x の値を示せ。

1.18.1 Q18 ブロックのソースコード

```
204   /* Q18 */
205   {
206     puts("Q18.実行した後の変数 x の値を示せ。");
207     puts("A18.変数 x の値は 0 である");
208     int x;
209     char *p;
210     p = "abcd";
211     if(p == "abcd") x = 0;
212     else x = 1;
213     printf("x ==> %x\n",x);
214   }
215
```

1.18.2 該当箇所の実行結果

```
Q18.実行した後の変数 x の値を示せ。
A18.変数 x の値は 0 である
x ==> 0
```

1.18.3 考察

a) if 文の条件式は文字列も扱うことができる.よって真となり,値は 0 となる.

1.19 (Q19)ポインタの代りに、 "int k;"を宣言し、配列を用いた文に書き換えよ。

1.19.1 Q19 ブロックのソースコード

```
216   /* Q19 */
217   {
218     puts("Q19.ポインタの代りに、 \"int k;\" を宣言し、配列を用いた文に書き換えよ。");
219     puts("A19.ソースコードの通りである");
220     int k;
221     char m[MAX];
222     for(k=0; m[k]; k++) m[k] += 1;
223   }
224
```

1.19.2 該当箇所の実行結果

```
Q19.ポインタの代りに、 "int k;" を宣言し、配列を用いた文に書き換えよ。
A19.ソースコードの通りである
```

1.19.3 考察

a) 書き換える前の文ではポインタ変数 p を宣言し,配列 m の先頭アドレスを格納している.
b) for 文ではポインタ変数 p を配列 m として扱っている.
c) 書き換えたあとの文では変数 k を宣言し,実際に配列自体を k によって操作している.

1.20 (Q20)次の定義による、 *q[2]、q[3][2]、*(q[2]+2)、*(*(q+3)+2)、**(q+1)の値を示せ。

1.20.1 Q20 ブロックと最後のソースコード

```
225     /* Q20 */
226     {
227       puts("Q20.次の値を示せ。");
228       puts("A20 それぞれの値は*q[2]=A、q[3][2]=7、*(q[2]+2)=C、*(*(q+3)+2)=7、**(q+1)=1");
229       static char *q[] = {"abcd", "12345", "ABCDEFG", "987"};
230       printf("*q[2]       ==> %c\n",*q[2]);
231       printf("q[3][2]     ==> %c\n",q[3][2]);
232       printf("*(q[2]+2)   ==> %c\n",*(q[2]+2));
233       printf("*(*(q+3)+2) ==> %c\n",*(*(q+3)+2));
234       printf("**(q+1)     ==> %c\n",**(q+1));
235     }
236
237
238     return(0);
239   }
```

1.20.2 該当箇所の実行結果

```
Q20.次の値を示せ。
A20 それぞれの値は*q[2]=A、q[3][2]=7、*(q[2]+2)=C、*(*(q+3)+2)=7、**(q+1)=1
*q[2]       ==> A
q[3][2]     ==> 7
*(q[2]+2)   ==> C
*(*(q+3)+2) ==> 7
**(q+1)     ==> 1
```

1.20.3 考察

a) *q[2]について.q[2]は 2 次元配列 q の 2 行目の先頭アドレスを表している.よって,値は A である
b) q[3][2]について.3 行目 2 列目の要素を示しているので,値は 7 である.
c) *(q[2]+2)について.2 行目の先頭アドレスに 2 を足しているので 2 列目を表している.よって,値は C である.これは,q[2][2]と同じ値である.
d) *(*(q+3)+2)について.2 次元配列 q の 3 行目の先頭アドレスを示した後, 2 を加えて 2 列目の要素を表している.よって値は 7 である.これは q[3][2]と同じ値である.
e) **(q+1)について.先頭アドレス q に 1 を加えているので q+1 は 1 行目の先頭アドレスを示している.よって値は 1 である.

２．リスト構造の考察

2.1 リスト構造プログラムのソースコード

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define FALSE 0
5   #define TRUE  !FALSE
6
7   typedef struct Node{
8     int num;
9     struct Node *next_ptr;
10  }node;
11
12  node *start_ptr = NULL;
13
14  void ins(int idata){
15    node *p = start_ptr;
16    printf("start_ptr ==> %08x\n",start_ptr);
17
18    start_ptr = (node *)malloc(sizeof(node));
19    if (start_ptr == NULL) puts("Not enough memory!"), exit(0);
20    printf("start_ptr ==> %08x\n",start_ptr);
21
22    start_ptr->num = idata;
23    start_ptr->next_ptr = p;
24    printf("&(start_ptr->num) ==> %08x\n",&(start_ptr->num));
25    printf("&(start_ptr->next_ptr) ==> %08x\n",&(start_ptr->next_ptr));
26  }
27
28  int main(){
29    int idata;
30    node *p;
31
32    puts("Enter a sequence of integers:");
33    while(scanf("%d", &idata) == TRUE) ins(idata);
34
35    puts("In reverse order:");
36    for(p = start_ptr; p != NULL; p = p->next_ptr){
37      printf("%5d-", p->num);
38      printf(":%08x\n", &(p->num));
39    }
40    puts("/end/");
41
42    return(0);
43  }
```

2.2 課題プログラムの実行結果

```
Enter a sequence of integers:
67
start_ptr ==> 00000000
start_ptr ==> e84000e0
&(start_ptr->num) ==> e84000e0
&(start_ptr->next_ptr) ==> e84000e8
46
start_ptr ==> e84000e0
start_ptr ==> e84039c0
&(start_ptr->num) ==> e84039c0
&(start_ptr->next_ptr) ==> e84039c8
8
start_ptr ==> e84039c0
start_ptr ==> e84039d0
&(start_ptr->num) ==> e84039d0
&(start_ptr->next_ptr) ==> e84039d8
1747
start_ptr ==> e84039d0
start_ptr ==> e84039e0
&(start_ptr->num) ==> e84039e0
&(start_ptr->next_ptr) ==> e84039e8
5555
start_ptr ==> e84039e0
start_ptr ==> e84039f0
&(start_ptr->num) ==> e84039f0
&(start_ptr->next_ptr) ==> e84039f8
In reverse order:
 5555-:e84039f0
 1747-:e84039e0
    8-:e84039d0
   46-:e84039c0
   67-:e84000e0
/end/
```

※scanf()関数には順番に 67,46,8,1747,5555 を入力.直後に Control+D を入力した.

2.3 考察

a) 7-10 行において構造体を宣言している.また,9 行目でメンバに自分自身のと同じタグの構造体をポインタで宣言している.これにより自己参照型構造体となり,リスト構造を作成することができる.

b) 7 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.

c) 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.

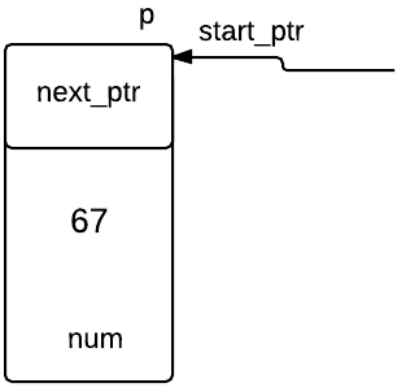
d) 12 行目において,リストに要素を入れる前はそれが空であるため NULL で初期化している.

e) 実行結果のアドレスを観察し,リスト構造の構成について次のような順序で新しい要素を付け加えてることがわかる.
   1. node 型ポインタ変数 p にリストの先頭アドレスを格納(15 行目)
   2. 新しい要素を,malloc 関数によりサイズを確保し作成する(18 行目)
   3. その要素に項目 num を格納する(22 行目)
   4. 1 つ前のリストの先頭アドレス(変数*p)にポイントさせる(23 行目)

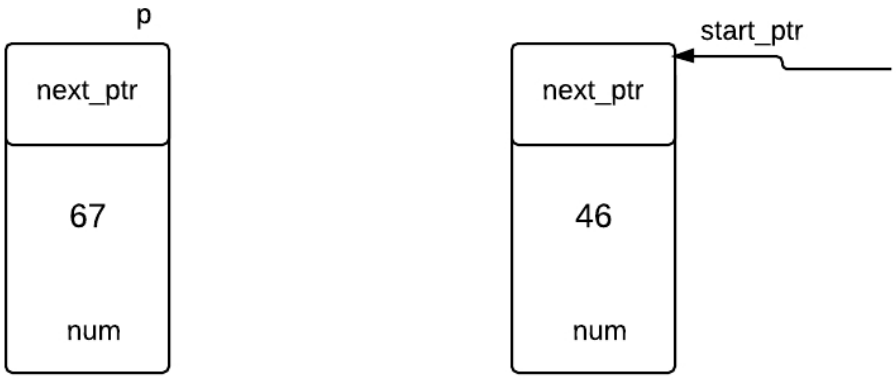
f) リストの要素を取り出す際,リスト構造の構成上,必然的に入力した順番と逆の順番で並んでいるのでその順で出てくる.

2.4 リスト構造の図的解釈

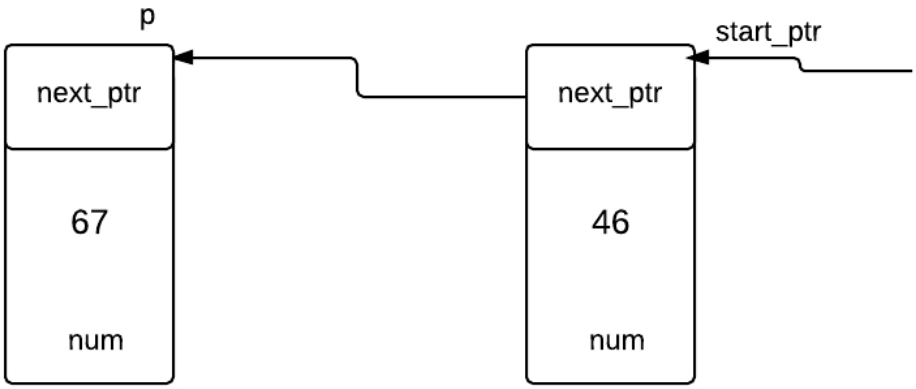
1.ポインタ変数 p にリストの先頭アドレスを格納



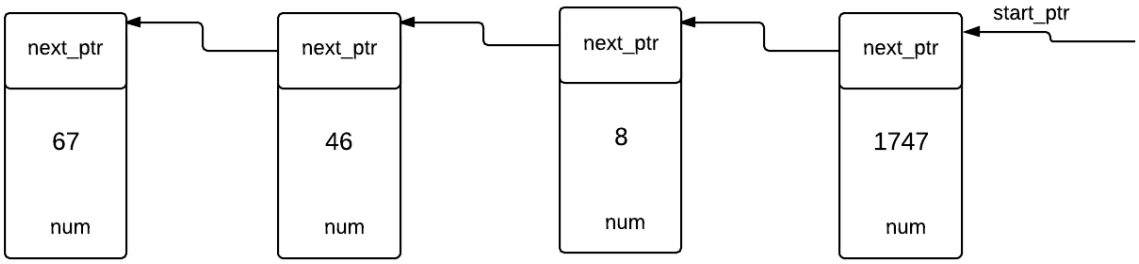
2.新しい要素を,malloc 関数によりサイズを確保し作成する
その要素に項目 num を格納する



3.1 つ前のリストの先頭アドレス(変数*p)にポイントさせる



4.以下繰り返す



--あとがき--

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

『初心者のためのポイント学習C言語-自己参照構造体』

http://www9.plala.or.jp/sgwr-t/c/sec15-5.html

2.反省・感想

試験の解答を確認するプログラムを200行書きました.自分はテストの方が良かったです.

ポインタの次に難しいとされる構造体について考えました.

次回のレポートのアルゴリズムの前哨戦と言ったところでしょうか…