プログラミングⅠ

report#7

提出日：7 月 11 日（木）

所属：琉球大学工学部情報工学科

学籍番号：135713B

氏名：天願 寛之

目次

1. 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。

1-a. 作成した解答確認プログラム

```
01 #include <stdio.h>
02
03 #define MAX 25 /*Q19 で使用する MAX の定義*/
04
05 int main(){
06 /* Q01 */ {
07    int v; /*int 型変数 v の宣言*/
08    printf("***Q1***¥n");
09    printf("&v = %08x¥n",&v); /*変数 v のアドレスを出力*/
10  }
11 /* Q2,Q3 */{
12    int m[5]; /*int 型配列 m の宣言*/
13    printf("¥n***Q2***¥n");
14    printf("&m[5] = %08x¥n",&m[5]); /*int 型配列 m の 5 番目のアドレスを出力(&m[5]で表示)*/
15    printf("m+5   = %08x¥n",m+5); /*int 型変数配列 m の 5 番目のアドレスを出力(m+5 で表示)*/
16    printf("***Q3***¥n");
17    printf("&m[0] = %08x¥n",&m[0]); /*int 型配列 m の先頭アドレスを出力(&m[0]で表示)*/
18    printf("m     = %08x¥n",m); /*int 型配列 m の先頭アドレスを出力(m で表示)*/
19  }
20 /* Q4 */{
21    int d[1][1]; /*int 型 2 次元配列 d の宣言*/
22    printf("¥n***Q4***¥n");
23    printf("&d[0][0] = %08x¥n",&d[0][0]); /*int 型 2 次元配列 d の先頭アドレスを出力
24 (&d[0][0]で表示)*/
25    printf("d[0]     = %08x¥n",d[0]); /*int 型 2 次元配列 d の先頭アドレスを出力(d[0]で表
26 示)*/
27    printf("*d       = %08x¥n",*d); /*int 型 2 次元配列 d の先頭アドレスを出力(*d で表示)*/
28  }
29 /* Q5 */{
30    int a=2, b=3, c=5, *p, *q; /*int 型変数 a,b,c を宣言し各々に値 2,3,5 を代入.また int 型
31 ポインタ変数 p,q を宣言*/
32
33    p = &b; /*ポインタ変数 p に b のアドレスを代入*/
34    q = &c; /*ポインタ変数 q に c のアドレスを代入*/
35    a = *p + *q; /*変数 a にポインタ p,q 要素の和の値を代入*/
36    printf("¥n***Q5***¥n");
37    printf("変数 a の値 = %d¥n",a); /*変数 a の値を出力*/
38  }
39 /* Q6 */{
40    int a=2, *p; /*int 型変数 a を宣言し、値 2 を代入.また、int 型ポインタ変数 p を宣言*/
41    p = &a; /*ポインタ変数 p に変数 a のアドレスを代入*/
42   *p = 5; /*ポインタ変数 p の値に 5 を代入*/
43    printf("¥n***Q6***¥n");
44    printf("変数 a の値 = %d¥n",a); /*変数 a の値を出力*/
45  }
46 /* Q7 */{
47    int a=200, b=300, *p, *q1, **q; /*int 型変数 a,b を宣言し、各々に値 200,300 を代入.ま 48 た、ポインタ変数 p,q1,2
重ポインタ変数 q の宣言*/
49
50
51    printf("¥n***Q7***¥n");
52    printf("アドレス&a の値  => ", &a);
53    printf(" a=%8d¥n", a); /*変数 a の値を出力*/
54    printf("アドレス&b ==> ", &b);
55    printf("&b=%08x  :アドレス 200 の代り¥n", &b); /*変数 b のアドレスを出力*/
56    printf("アドレス&b の値  => ", &b);
57    printf(" b=%8d¥n", b); /*変数 b の値を出力*/
```

```
58
59    p  = &a; /*ポインタ変数 p に変数 a のアドレスを代入*/
60     q1 = &b; /*ポインタ変数 q1 に変数 b のアドレスを代入*/
61     q  = &q1; /*2 重ポインタ変数 q にポインタ変数 q1 のアドレスを代入*/
62     printf("*p=%d¥n", *p); /*ポインタ変数 p の指すアドレスの内容を出力*/
63     printf("*q=%08x   :アドレス 200 の代りと同じ！ ¥n", *q); /*2 重ポインタ変数 q の指すア 64 ドレスを出力(問題では
200)*/
65     printf("**q=%d¥n", **q); /*2 重ポインタ変数 q の指すアドレスの指すアドレスの内容を出力 66 */
67     puts("omake");
68     printf(" q=%08x   :アドレス 100 と考える¥n", q); /*2 重ポインタ変数 q の値を出力(問題で 69 は 300)*/
70   }
71 /* Q8 */{
72     int m[]={1,2,3,4,5}, k; /*int 型配列 m を宣言し,1,2,3,4,5 を代入.また int 型変数 k の宣 73 言*/
74     printf("¥n***Q8***¥n");
75     for(k=0; k<5; k++){ /*for ループ*/
76       printf("k      = %d¥n",k); /*変数 k の値を出力*/
77       printf("m[k]   = %d¥n",m[k]); /*配列の要素を出力(m[k]で表示)*/
78       printf("*(m+k) = %d¥n¥n",*(m+k)); /*配列の要素を出力(*(m+k)で表示)*/
79     }
80   }
81 /* Q9 */{
82     int *p, q; /*int 型変数 q と int 型ポインタ変数 p の宣言*/
83     p = &q; /*ポインタ変数 p に変数 q のアドレスを代入*/
84     printf("¥n***Q9***¥n");
85     printf("p   = %08x¥n",p); /*ポインタ変数 p の値を出力*/
86     printf("p+1 = %08x¥n",p+1); /*p+1 を出力*/
87     printf("p+2 = %08x¥n",p+2); /*p+2 を出力*/
88   }
89 /* Q10 */{
90     int m[]={1,2,3,4,5}, *p; /*int 型配列 m を宣言し、1,2,3,4,5 を代入.int 型ポインタ変数 91 p を宣言*/
92     printf("¥n***Q10***¥na.¥n");
93     printf("m[0] = %d¥n",m[0]); /*配列 m の先頭の要素を出力(m[0]で表示)*/
94     printf("*m   = %d¥n",*m); /*配列 m の先頭の要素を出力(*m で表示)*/
95     printf("b.¥n");
96     p = m; /*ポインタ変数 p に配列 m を代入*/
97     printf("*p   = %d¥n",*p); /*ポインタ変数 p の先頭の要素を出力(*p で表示)*/
98     printf("p[0] = %d¥n",p[0]); /*ポインタ変数 p の先頭の要素を出力(p[0]で表示)*/
99   }
100 /* Q11 */{
101     static int m[5] = {10, 20, 40, 50, 30}; /*静的な int 型配列 m を宣言し、値 10,20,40,50,301
102 を代入*/
103     printf("¥n***Q11***¥n");
104     printf("*m        = %d¥n",*m); /*配列 m の先頭要素を出力*/
105     printf("*(m+3)    = %d¥n",*(m+3)); /*(m+3)の要素を出力*/
106     printf("*m+3      = %d¥n",*m+3); /*配列 m の先頭要素に 3 を足して、その要素を出力*/
107     printf("*m+*(m+3) = %d¥n",*m+*(m+3)); /*配列 m の先頭要素と*(m+3)の要素の和を出力*/
108   }
109 /* Q12 */{
110     static int d[][3] = {{1,2,3}, {5,6,7}, {4,6,8}, {9,7,5}}; /*静的な int 型 2 重配列 111 d を宣言し、値を代
入*/
112    printf("¥n***Q12***¥n");
113    printf("*d[2]     = %d¥n",*d[2]); /**d[2]の要素を出力.2 重配列では d[2][0]*/
114   printf("*(d[2]+2) = %d¥n",*(d[2]+2)); /**(d[2]+2))の要素を出力.2 重配列では d[2][2]*/
115    printf("*d[2]+2   = %d¥n",*d[2]+2); /**d[2]+2 の要素を出力.2 重配列 d[2][0]の要素+2*/
116    printf("**d       = %d¥n",**d); /***d を出力.2 重配列では d[0][0]*/
117    printf("*(*d+3)   = %d¥n",*(*d+3)); /**(*d+3)を出力.2 重配列では d[1][0]*/
118    printf("**d+6     = %d¥n",**d+6); /***d+6 を出力.2 重配列 d[0][0]の要素+6*/
119    printf("*(d[1]+2) = %d¥n",*(d[1]+2)); /**(d[1]+2)を出力.2 重配列では d[1][2]*/
120    printf("**(d+2)   = %d¥n",**(d+2)); /***(d+2)を出力.2 重配列では d[0][2]*/
121  }
122 /* Q13 */{
123    char *str = "abcdefg", *p; /*char 型ポインタ変数 str,p を宣言し、ポインタ変数 str に文 124 字列を代入。*/
125    p = str + 3; /*ポインタ変数 p に str + 3 を代入*/
```

```
126    printf("¥n***Q13***¥n");
127    printf("p = %s¥n",p); /*p の値を出力*/
128  }
129 /* Q14 */{
130    char *p; /*char 型ポインタ変数 p を宣言*/
131    p = "abc"; /*ポインタ変数 p に文字列を代入*/
132    printf("¥n***Q14***¥n");
133    printf("&(*p)  = %08x(アドレス 100 の代わり)¥n",&(*p)); /*(*p)のアドレスを出力(問題 134 では 100)*/
135    printf("p      = %s¥n",p); /*ポインタ変数 p の値を出力*/
136    printf("*p     = %c¥n",*p); /*p の指すアドレスの内容を出力*/
137    printf("*(p+2) = %c¥n",*(p+2)); /*(p+2)の指すアドレスの内容を出力*/
138  }
139 /* Q15 */{
140    static char m[] = "abcd"; /*静的な char 型配列 m を宣言し、文字列を代入*/
141    char *p, *q; /*char 型ポインタ変数 p,q を宣言*/
142    p = &m[0]; /*ポインタ変数 p に配列 m の先頭アドレスを代入*/
143    q = m; /*ポインタ変数 q に配列 m の先頭アドレスを代入*/
144    printf("¥n***Q15***¥n");
145    printf("*m = %c¥n",*m); /*配列 m の指すアドレスの要素を出力*/
146    printf("*p = %c¥n",*p); /*ポインタ変数 p の指す先頭アドレスの要素を出力*/
147    printf("*q = %c¥n",*q); /*ポインタ変数 q の指す先頭アドレスの要素を出力*/
148  }
149 /* Q16 */{
150    static char m[] = "abcd"; /*静的な char 型配列 m を宣言し、文字列を代入*/
151    char *p; /*char 型ポインタ変数 p を宣言*/
152    p = &m[2]; /*ポインタ変数 p に m[2]のアドレスを代入*/
153    printf("¥n***Q16***¥n");
154    printf("*p     = %c¥n",*p); /*ポインタ変数 p の指す先頭アドレスの要素を出力*/
155    printf("*(m+2) = %c¥n",*(m+2)); /*(m+2)を出力(配列では m[2])*/
156    printf("*m+2   = %c¥n",*m+2); /**m+2 を出力(配列では m[0]+2)*/
157  }
158 /* Q17 */{
159    char p[] = "abcd"; /*char *p; p = "abcd";と同価値*//*char 型配列 p を宣言し、文字列 160 を代入*/
161    *(p + 1) = 'x'; /*(p+1)の内容に x を代入*/
162    printf("¥n***Q17***¥n");
163    printf("p = %s¥n",p); /*配列 p の値を出力*/
164  }
165 /* Q18 */{
166    int x; /*int 型変数 x を宣言*/
167    char *p; /*char 型ポインタ変数 p を宣言*/
168    p = "abcd"; /*ポインタ変数 p に文字列を代入*/
169    if(p == "abcd") x = 0; /*if 文.p=="abcd"であれば x に 0 を代入、それ以外は x に 1 を代入 170 */
171    else            x = 1;
172    printf("¥n***Q18***¥n");
173    printf("x = %d¥n",x); /*変数 x の値を出力*/
174  }
175 /* Q19 */{
176    int k; /*int 型変数 k を宣言*/
177    char m[MAX]; /*char 型配列 m を宣言*/
178    for(k=0; m[k]; k++) m[k] += 1; /*for 文.初期値 k=0.条件 m[k].ループ毎の動作 m[k] +=1*/
179    printf("¥n***Q19***¥n");
180    printf("解答の明記¥nint k;¥nchar m[MAX];¥nfor(k=0; m[k]; k++) m[k] += 1;¥n");
181  }
182 /* Q20 */{
183    static char *q[] = {"abcd", "12345", "ABCDEFG", "987"}; /*静的な char 型ポインタ配 184 列 q を宣言し、文字
列を代入*/
185
186    printf("¥n***Q20***¥n");
187    printf("*q[2] = %c¥n",*q[2]); /**q[2]を出力.ポインタ配列 q では q[2][0]の要素*/
188    printf("q[3][2] = %c¥n",q[3][2]); /*q[3][2]の要素を出力*/
189    printf("*(q[2]+2) = %c¥n",*(q[2]+2)); /**(q[2]+2)を出力.ポインタ配列 q では q[2][2]
190 の要素*/
191    printf("*(*(q+3)+2) = %c¥n",*(*(q+3)+2)); /**(*(q+3)+2)を出力.ポインタ配列 q では 192 q[3][2]の要素*/
```

```
193    printf("**(q+1) = %c¥n",**(q+1)); /***(q+1)を出力.ポインタ配列 q では q[1][0]の要素 194 */
195  }
196  return(0);
197 }
```

1-b. 出力結果

```
***Q1***
&v = 52404b50

***Q2***
&m[5] = 52404b44
m+5   = 52404b44
***Q3***
&m[0] = 52404b30
m     = 52404b30

***Q4***
&d[0][0] = 52404b2c
d[0]     = 52404b2c
*d       = 52404b2c

***Q5***
変数 a の値  = 8

***Q6***
変数 a の値  = 5

***Q7***
アドレス&a の値  =>  a=     200
アドレス&b     ==> &b=52404af8   : アドレス 200 の代り
アドレス&b の値  =>  b=     300
*p=200
*q=52404af8
**q=300
omake
 q=52404ae0

***Q8***
k      = 0
m[k]   = 1
*(m+k) = 1

k      = 1
m[k]   = 2
*(m+k) = 2

k      = 2
m[k]   = 3
*(m+k) = 3

k      = 3
m[k]   = 4
*(m+k) = 4

k      = 4
m[k]   = 5
*(m+k) = 5


***Q9***
p   = 52404aac
```

```
p+1 = 52404ab0
p+2 = 52404ab4

***Q10***
a.
m[0] = 1
*m   = 1
b.
*p   = 1
p[0] = 1

***Q11***
*m       = 10
*(m+3)   = 50
*m+3     = 13
*m+*(m+3) = 60

***Q12***
*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

***Q13***
p = defg

***Q14***
&(*p)  = 0d7fcdb6(アドレス100の代わり)
p      = abc
*p     = a
*(p+2) = c

***Q15***
*m = a
*p = a
*q = a

***Q16***
*p     = c
*(m+2) = c
*m+2   = c

***Q17***
p = axcd

***Q18***
x = 0

***Q19***
int k;
char m[MAX];
for(k=0; m[k]; k++) m[k] += 1;

***Q20***
*q[2] = A
q[3][2] = 7
*(q[2]+2) = C
*(*(q+3)+2) = 7
**(q+1) = 1
```

1-c. 解説 ※コメントでは説明できなかった部分を説明する。

イ. 1-a のプログラムは資料「問題と解答」の「II ポインタとアドレス」より、各問題ごとの解答確認プログラムとして作成したものである。

ロ. このプログラムは複数のブロック文を用いることで、その中だけで有効な変数を宣言し、他のブロック文への影響を無くしている。そうすることで 1 つのプログラムで複数の動作を見る事が出来る。

ハ. 30 行目のようにポインタ変数の宣言は[データ型 *変数名]と記述する。

ニ. 183 行目のようにポインタ配列の宣言は[データ型 *変数名[大きさ]]と記述する。

ホ. 21 行目のように 2 次元配列の宣言は[データ型 変数名[大きさ][大きさ]]と記述する。

ヘ. 47 行目のように 2 重ポインタ変数の宣言は[データ型 **変数名]と記述する。

ト. 33,34 行目のように[ポインタ変数 = &変数]と記述することでポインタ変数に変数のアドレスを代入することができ、9 行目のように'&変数'を%x(16 進数)で出力すると、その変数の先頭アドレスが表示される

チ. 35 行目のように[変数 = *ポインタ変数](ここでの"*"は間接参照詞という)と記述することでポインタ変数が格納しているアドレスが持つ内容を代入することができ、また 62 や 146 行目のように"*ポインタ変数"を%d(10 進数)や%c(文字)で出力すると、そのポインタ変数が格納している先頭アドレスの値が表示される。

リ. 61 行目のように[2 重ポインタ変数 = &ポインタ変数]と記述することでポインタ変数のアドレスを 2 重ポインタ変数に代入することができ、65 行目や 193 行目のように"**2 重ポインタ変数"を%d や%c で出力すると、その 2 重ポインタ変数が格納しているポインタの格納している先頭アドレスが持つ値が表示される。

ヌ. 101 行目のように"statics"は記憶域クラス指定子の静的変数で、コンパイル時にその変数値記憶用のメモリ領域を、固定的に確保してしまう。

1-d. 考察

イ. Q2,Q3 より配列 m を 16 進数で出力すると配列の先頭アドレスが表示され、m+5 をして 16 進数で出力すると 5 番目の配列のアドレスが表示されていることが分かる。

ロ. Q4 より”&d[0][0]”、”d[0]”、”*d”の出力は同じアドレスを示している。

ハ. Q5 よりポインタ変数 p,q に変数 b,c のアドレスを代入し、そのポインタ変数の指すアドレスの内容を足した事で変数 a には 8 が代入されている。

ニ. Q6 より、ポインタ変数 p に変数 a のアドレスを代入し、その内容に 5 を足したことで 5 が出力されている。

ホ. Q7 は出力から分かるようにアドレスを 100,200 というのは無理なので int 型変数を宣言し 200,300 を代入。その変数のアドレスをポインタ変数 p,q1 に代入し、2 重ポインタ変数 q に p1 のアドレスを代入。よって”*p”,”**q”の出力は 200,300。”*q”の出力は変数”b”のアドレス(これはアドレス 200 の代わり)となっている。

ヘ. Q8 より m[k]と*(m+k)の出力は等しい。配列名の出力は配列の先頭アドレスを指し、配列はアドレスが並んでいることから m+k とすることで m のアドレスを k 分ズラし k 番目の値を出力している。

ト. Q9 より p と p+2 のアドレスの差は 8 バイトとわかる。int 型は 4 バイトなので、ポインタの演算はデータ型に依存していることが分かる。

チ. Q10 より a,b はどちらも正しいことが分かる。「考察、ヘ. Q8」でも述べているとうり、配列名は先頭アドレスを指しているのでポインタと同じ書式で使用出来る。逆にポインタ変数を配列名と同じ書式で使用しても良い。

リ. Q11 より*(m+3)は「考察、ヘ. Q8」でも述べているとうり m+3 とすることで 3 番目の配列要素を出力している。*m+3 は先頭要素に 3 を足した値となる。*m+*(m+3)は*m と*(m+3)の和である。

ヌ. Q12 より 2 次元配列 d を宣言したもので、2 番目の出力”*(d[2]+2)”は 2 次元配列の”d[2][]”の後ろの方からアドレスを足していき、その要素を出力している。つまり、コメントにもあるように”d[2][2]”の出力をする。3 番目の出力は 1 番目の出力要素に 2 を加算したものである。4 番目の出力”**d”は 2 重配列が 2 重ポインタと同じ意味をなすことから最初の要素を示している。

ル. Q13 よりポインタ変数 p に代入するポインタ変数 str の値は文字列の先頭アドレスなので 3 を加算することで p の指すアドレスを 3 つズラしている。したがって、文字列はアドレスが並んでいるので、p の出力は文字列の 4 番目のアドレスからの出力になる。

ヲ. Q14 より&(*p)のアドレスは 100 に限らないので仮のアドレスである。*p の出力は文字列の先頭アドレスの要素である。*(p+2)の出力は文字列の先頭アドレスから 2 つズラしたアドレスの要素を出力している。

ワ. Q15 より、142,143 行目は同じようにポインタ変数 p,q に配列 m の先頭アドレスを代入している。してがって、同じ値を出力する。

カ. Q16 より、ポインタ変数 p には配列 m の 3 番目のアドレスを代入している。よって*p の出力は 3 番目の要素。*(m+2)は先頭アドレスから 2 つズラしたアドレスの要素を出力。*m+2 は先頭要素 a に 2 を加算した値を出力。よって 3 つとも同じ c の値を出力する。

ヨ. Q17 より 161 行目では配列 p の先頭アドレスから 1 つズラした要素に x を代入している。したがって、p の出力では 2 番目の要素が x に変わってしまう。

タ. Q18 より if 文を用いてポインタ変数 p の値が”abcd”であれば x に 0 を代入し、それ以外では 1 を代入している。コンパイラによって出力が異なることもある。

レ. Q19 について以下が元のコードである

```
char m[MAX], *p;

for(p=m; *p; ++p) *p += 1;
```

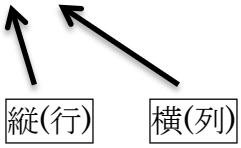
上記では初期値を配列 m の先頭アドレス、条件式をポインタ変数 p の値の間、ループ毎の操作としてポインタ変数のアドレスをインクリメントしている。さらに for 文が真である間ポインタ変数 p の要素に 1 を加算している。Q19 ではポインタ変数 p に代わり、変数 k を宣言し、初期値を k=0、条件式を配列 m の k 番目の要素の間、ループ毎の操作として変数 k をインクリメントしている。そして for 文が真である間、配列 m の k 番目の要素に 1 を加算している。

ソ. Q20 より”*(q[2]+2)”の出力は 2 重ポインタの q[2][0]に 2 を加算したのと同じで、後ろの方から加算される。よってその出力は q[2][2]の出力になる。”*(*(q+3)+2)”の出力は、まずポインタ配列 q に 3 を加算し q[3][0]を示し、その値に 2 を加算する事で q[3][2]を示し、その値を出力している。”**(q+1)”の出力はポインタ配列 q に 1 を加算する事で q[1][0]を示し、その値を出力している。ここで言える事はポインタ配列では、q[0]に加算すると 2 次元配列での q[0][加算した値]というように後ろの方から加算され、q に加算すると 2 次元配列での q[加算した値][0]というように加算される。またポインタ配列は 2 重ポインタ、2 次元配列と同じ意味を持ってくる。

1-e. 2 次元配列について

型、配列名、縦(行)と横(列)の大きさを宣言します。

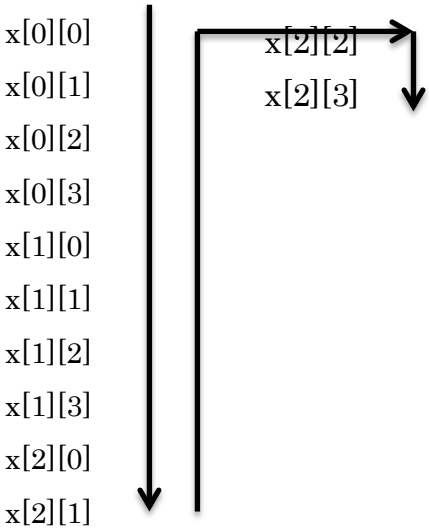
int x[3][4];



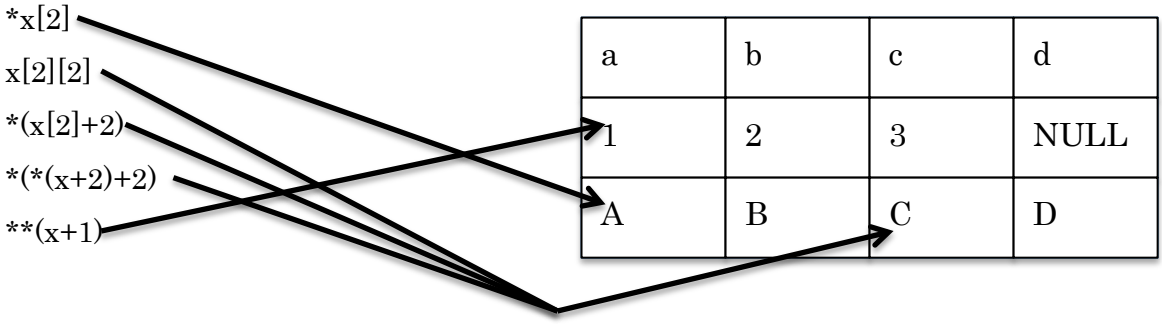
概念図 →

x[0][0]	x[0][1]	x[0][2]	x[0][3]
x[1][0]	x[1][1]	x[1][2]	x[1][3]
x[2][0]	x[2][1]	x[2][2]	x[2][3]

メモリ上の並び(アドレスは並んでいる)



int x[3][4] = {“abcd”,”123”,”ABCD”};と記述したときの概念図



a	b	c	d
1	2	3	NULL
A	B	C	D

2. 構造体について

2-a. **構造体とはC言語**などのプログラミング言語が持つデータ型の一つで、複数の異なるデータ型の変数を一つにまとめたものである。構造体を扱うには、まず構造体の型の定義(Cでは“struct”というキーワードを用いる)を記述し、その型の変数として構造体の実体を宣言する。構造体を構成する要素を「メンバ」(member)あるいは「メンバ変数」などと呼ぶ。

2-b. ソースコード

```
01 #include <stdio.h>
02 /* （1）構造体の型枠の宣言 */
03 struct account {
04   int namber;        /* 学生番号 */
05   char name[20];  /* 氏名 */
06   double average; /* なんかの平均 */
07 };
08
09 int main(void){
10   int i;
11   /* （2）構造体の宣言 */
12   /* （3）構造体の初期化 */
13   struct account student1 = { 1, "Jana Yuuta", 99.9 };
14   struct account student2[20] = {
15     { 2, "Goya Minato", 88.8 },
16     { 3, "Touyama Yuuta", 77.7 },
17     { 4, "Higasionna Takui", 66.6 },
18   };
19   /* （4）構造体の参照 */
20   printf("%d %s %5.1f¥n¥n",student1.namber,student1.name,student1.average);
21  for(i = 0; i < 3; i++) {
22   printf("%d %s %5.1f¥n",
23           student2[i].namber, student2[i].name, student2[i].average);
24   }
25  return (0);
26 }
```

2-c. 実行結果

```
1 Jana Yuuta  99.9


2 Goya Minato  88.8
3 Touyama Yuuta  77.7
4 Higasionna Takui  66.6
```

2-d. 解析

イ. このプログラムは構造体accountを定義し、宣言。構造体の各変数に学生番号、氏名、何かの平均を代入したプログラムである。

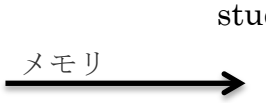
ロ. 02行目(1)構造体の型枠の宣言について、まず複数のデータ(メンバ)をまとめて、1つの構造体の型枠を宣言している。この型枠の有効範囲は宣言する場所によって異なり、この関数外で宣言した場合はその位置より下の全関数で有効となる。また、関数内で宣言した場合は宣言した関数内でのみ有効となる。
記述の仕方`struct 構造体名{ データ型　メンバ名 ;};`

ハ. 11行目(2)構造体の宣言について、(1)で作った型枠を使って実際にデータを宣言し、メモリ上に領域を確保している。構造体は構造体変数として1つの構造体を扱うこともでき、複数の構造体をまとめて構造体配列として扱うことも出来る。
記述の仕方`struct 構造体名　変数名の並び;`

構造体変数の要領の確保の例

```
struct account student1;
```

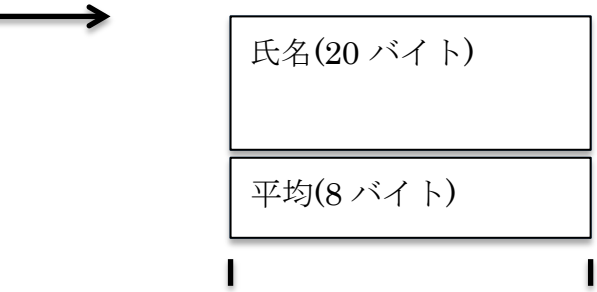
メモリ

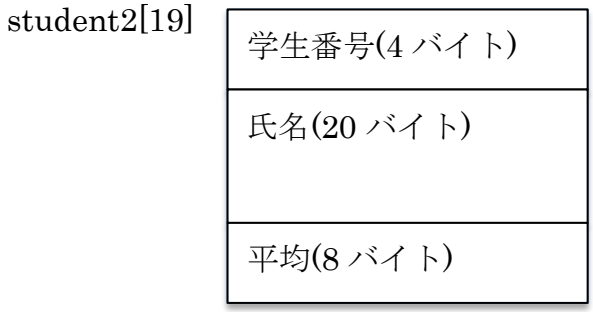


構造体配列の要領の確保の例

```
struct account student2[20];
```

メモリ





ハ. 12行目(3)構造体の初期化について、{}の間に各メンバ名をカンマで区切って記述する。

```
struct account student1 = { 1, "Jana Yuuta", 99.9 };
```

構造体配列の初期化について、各配列要素ごとに{}で区切って記述する。

```
struct account student2[20] = {
    { 2, "Goya Minato", 88.8 },
    { 3, "Touyama Yuuta", 77.7 },
    { 4, "Higasionna Takui", 66.6 },
  };
```

ニ. 19行目(4)構造体の参照について、構造体の各メンバは、[構造体変数名. メンバ名]のようにピリオドを用いて指定する。このピリオドはドット演算子と呼ぶ。

構造体変数の参照

```
printf("%d %s %5.1f¥n¥n",student1.namber,student1.name,student1.average);
```

構造体配列の参照

```
for(i = 0; i < 3; i++) {
  printf("%d %s %5.1f¥n",
          student2[i].namber, student2[i].name, student2[i].average);
  }
```

ホ. 構造体変数や構造体配列は通常の変数や配列と同様に扱えるので記憶クラスも指定する事が出来る。

2-e. 考察

イ. 構造体を用いる事で複数のデータ型を扱う事が出来る。

3. リスト構造について

リスト構造は、構造体のデータをポインタで接続したデータ構造。ポインタの付け替えで、構造体データの参照、追加や削除が可能。

3-a. ソースコード

```
/*
  Program    : list.c
  Comment    : リスト構造
*/

#include <stdio.h>
#include <stdlib.h>

#define FALSE 0
#define TRUE  !FALSE

/*自己参照構造体の宣言*/
typedef struct Node{
  int num; /*int 型変数 num の宣言*/
  struct Node *next_ptr; /*struct 型のデータを指すポインタ変数 next_ptr を宣言*/
}node; /*struct Node を型名 node と定義*/

node *start_ptr = NULL; /*node 型のグローバルポインタ変数を宣言し NULL を代入*/

/*void 型関数 ins の定義,int 型引数 idata をとる*/
void ins(int idata){
  node *p = start_ptr; /*node 型のポインタ p に start_ptr を代入*/

  start_ptr = (node *)malloc(sizeof(node)); /*start_ptr に(node *)malloc(sizeof(node))
を代入*/
  if (start_ptr == NULL) puts("Not enough memory!"), exit(0); /*if 文でメモリ要領の判
  定*/
  start_ptr->num = idata; /*ポインタ型構造体変数からメンバへのアクセス及び int 型引数 idata
  を代入*/
  start_ptr->next_ptr = p; /*ポインタ型構造体変数からメンバへのアクセス及び node 型ポイン
  タ p を代入*
  printf("%d¥nstart_ptr***%08x***¥n",sizeof(node),start_ptr);
  printf(“next_ptr***%08***¥n¥n”,start_ptr->next_ptr);}
  }
int main(){
  int idata; /*int 型変数 idata の宣言*/
  node *p; /*node 型ポインタ変数 p の宣言*/

  puts("Enter a sequence of integers:");
  /*while ループ,真ならば関数 ins を引数 idata として呼び出し*/
  while(scanf("%d", &idata) == TRUE) ins(idata);

  puts("In reverse order:");
  for(p = start_ptr; p != NULL; p = p->next_ptr){ /*for ループ*/
    printf("%5d-", p->num);  /* アドレスと値の出力へ更新しなさい */

    printf("¥tstart_ptr***%08x***",p);
    printf("¥tnext_ptr***%08x***¥n",p->next_ptr);
  }
  puts("/end/");

  return(0);
}
```

3-b. 実行結果(入力されたデータは1,12,123である)

```
Enter a sequence of integers:
1
確保した要領の大きさ16
nowアドレス***10c000e0***
nextアドレス ***00000000

12
確保した要領の大きさ16
nowアドレス***10c03a00***
nextアドレス ***10c000e0

123
確保した要領の大きさ16
nowアドレス***10c03a10***
nextアドレス ***10c03a00

.
In reverse order:
  123-       現在のアドレス***10c03a10***          nextアドレス***10c03a00***
   12-       現在のアドレス***10c03a00***          nextアドレス***10c000e0***
    1-       現在のアドレス***10c000e0***          nextアドレス***00000000***
/end/
```

3-c. 解析

イ. 15行目で構造体のメンバの中に構造体と同じ型のデータを指し示すポインタのメンバを持っている。このように構造体自身の型へのポインタのメンバをもつ形式の構造体を自己参照構造体と呼ぶ。

ロ. 13行目のように”typedif struct Node{}node;”と記述することで以後struct Nodeをnodeとして定義し、構造体変数の宣言で扱える。

ハ. 41行目でwhileループ、入力したデータが真であれば関数insを引数idataを渡して呼び出す。偽であれば43行目以降を実行。

ニ. 44行目でforループ、node型のポインタpの初期値start_ptr、条件式NULLが出るまで、ループ毎の動作node型のポインタpで構造体のデータを指すポインタnext_ptrにアクセスし、そのアドレスをpに代入。

ホ. 24行目の”malloc”はプログラムの走行中に必要なメモリ領域を動的に確保する事が出来る。malloc()関数の引数は、確保する領域の大きさで24行目の引数”(sizeof(node))”では構造体のメモリ領域を確保している。また、左辺と右辺の型を合わせるために”(node*)”とすることで、node型のポインタにキャストしている。

ヘ. 27行目のif文でstart_ptrがNULLを指せば"Not enough memory!"を出力し、抜け出す。

ト. 28行目や30行目のアロー演算子(->)はメンバへのアクセスするために用いられる。

チ. 28行目ではポインタ型構造体変数start_ptrからメンバnumへアロー演算子(->)を用いてのアクセス及びint型引数idataを代入することでnumにデータが格納される。

リ. 30行目ではポインタ型構造体変数start_ptrからメンバnext_ptrへ演算子(->)を用いてのアクセス及びnode型ポインタpを代入することでnext_ptrにpの値(ここではNULL)を入れる。

ヌ. 18行目でグローバルなポインタ型構造体変数start_ptrを宣言し、NULLのアドレス(アドレスをもっていないので値は0)を代入している。22行目でポインタ型構造体変数start_ptrのアドレスをポインタ型構造体変数prevに代入し、24行目でmalloc関数で確保したメモリの容量の先頭アドレスをstart_ptrに代入している。

3-d. 考察

イ. 出力結果よりmalloc関数で確保したメモリの容量は16バイトである。

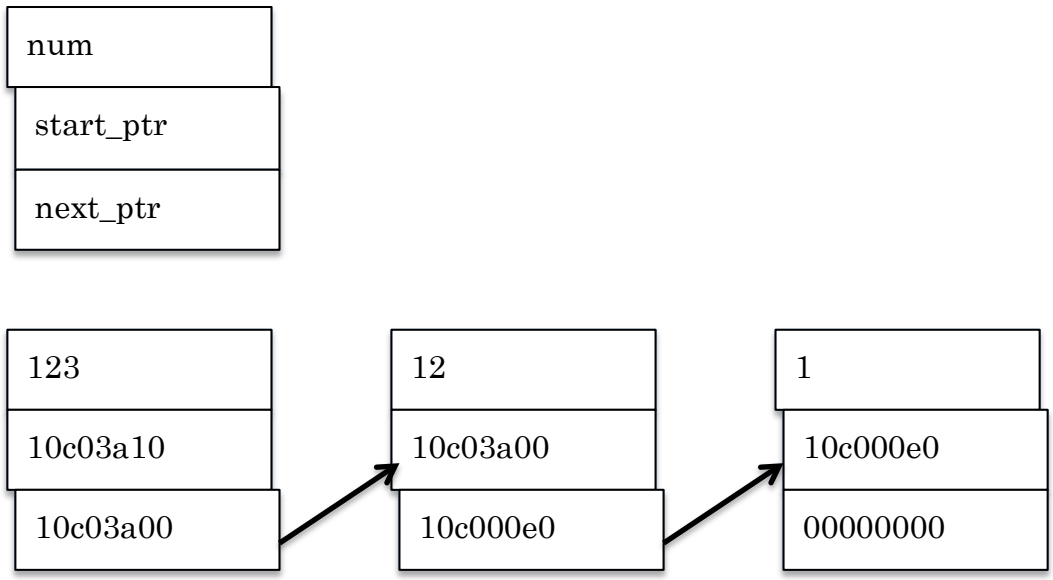
ロ. 再度異なるデータが入力されると再度同じメモリの領域を確保している。

ハ. 出力結果よりコマンド入力した値のアドレスは入力した順に小さく、入力の逆順に次のデータのアドレスを指している。最初に入力したデータは次のデータのアドレスを指す事が出来ないのでアドレスの出力はNULL。つまりアドレスが無い事を示している。

ニ. while文の一回目の動作によるins関数では次に続くデータが無いので同じ構造体の中にある、次のデータへのポインタはNULLとなっている。

リスト構造体の概念図

(numは入力されたデータ、start_ptrはデータのアドレス、next_ptrはデータの指す次のアドレス)



4. 感想

問題の解答を確認するプログラムを作成することとその考察にはとても時間がかかりました。なんとかリスト構造も構造体と合わせ考察しましたが、とても難しかったです。次の課題も気を引き締めて取り組みたいです。