

補講 (中間試験不合格者向け)

山本昌志*

2005 年 8 月 23 日

本テキストは、中間テストでの不合格者に向けての補講に使用する。補講対象者は、このテキストに書いてある内容を理解し、前回の中間試験の問題が全て解けるようになって、再試験に臨むこと。

1 いろいろなデータ構造

これまで 3 つのデータ構造を学習した。単純型と配列、構造体である。それぞれの概略は以下の通り。

1.1 単純型

単純型の変数は、次のように変数に一つの数値¹しか代入できないものを言う。これを使う場合、以下のよう宣言する。

```
char c, h, moji;  
int i, j, seisu;  
double x, y, jisu;
```

このイメージは、図 2 に示しているとおりで、変数とは数値を入れる箱のようなものである。整数型と倍精度実数型の変数は、数学の変数とよく似ている。

| | | | |
|--------|--|---|--|
| char | <div><div>y</div><div>c</div></div> | <div><div>a</div><div>h</div></div> | <div><div>A</div><div>moji</div></div> |
| int | <div><div>54878974</div><div>i</div></div> | <div><div>-9637</div><div>j</div></div> | <div><div>635</div><div>seisu</div></div> |
| double | <div><div>3.141592</div><div>x</div></div> | <div><div>1.63585e+56</div><div>y</div></div> | <div><div>-5.9684e-26</div><div>jisu</div></div> |

図 1: 単純型のイメージ。変数とはデータを入れる箱のようなもの。

*国立秋田工業高等専門学校 電気情報工学科

¹一つの文字のみ代入可能なものは文字型の変数である。文字も整数として扱えるので、この範疇と考える。

1.2 配列

順序づけられた同じ型のデータが複数ある場合、配列の出番となる。添え字 (これが順序を表す) により、それらにアクセスできるので、データの操作が簡単にできる。配列を使う場合、

```
int i[10], j[100][100];
```

のように宣言を行う。そうすると必要なメモリー領域が確保され、配列が使えるようになる。この配列のデータにアクセスするためには、配列名と添え字を指定する。次のようにである。

```
i[3]=5;  
c=i[3];
```

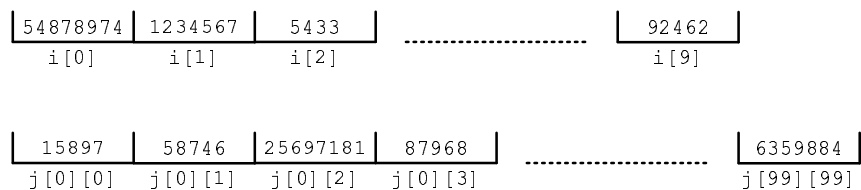


図 2: 配列のイメージ。データを入れる箱がいっぱいある。ただし箱の大きさは全て同じ。

1.3 構造体

1.3.1 構造体の概要

配列は同じ型のデータの集まりであったが、構造体は異なる型のデータの集まりである²。この構造体を使う場合、

1. 構造体のメンバーを規定する。これにより構造体を定義する。
2. 構造体変数の宣言。これによりメモリーが確保される。

という手順が必要である。このようにして、構造体を定義し、メモリーを確保した後、それを使うことができる。今までは、データの型の内容があらかじめ決まっていたので、最初の手順は不要であった。一方、構造体のメンバー、すなわちデータの型はプログラマーが決めなくてはならないので、最初の手順が必要となる。

最初のメンバーの規定は、つぎのように行う。

```
struct seito{  
    char name[128];  
    int kokugo;  
    int sansu;  
};
```

²同じ型でも良い

ここでは、メモリーがまだ確保されていないことに注意が必要である。これは、プログラマーが新たに変数を定義したのと同じである。

そして、この構造体を実際に使う場合には、次のようにしてメモリーを確保する。

```
struct seito yamamoto, ninensei[50];
```

そうすると構造体変数が使用可能となる。

メモリーに確保された構造体のデータにアクセスするためには、ドット (.) 演算子を使うことになる。次のようにである。

```
yamamoto.sansu = 63;
ninensei[5].kokugo=76;
tensu = ninensei[6].sansu;
```

1.3.2 構造体のイメージ

構造体を図に示すと図 3 の用になる。この場合構造体は

```
struct kouzoutai{
    char name[8];
    int seisu;
    double jisu;
};
```

の用に定義し、

```
struct kouzoutai a, b[10];
```

のように構造体変数を用意する。

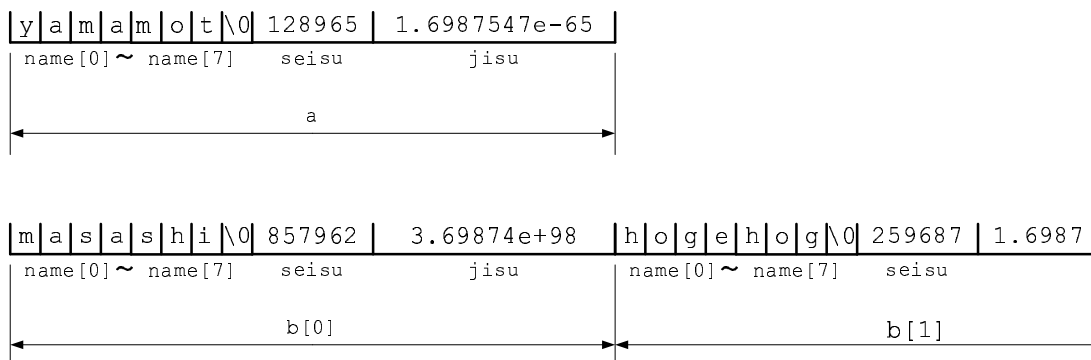


図 3: 構造体のイメージ。データを入れるいろいろな大きさの箱がある。

1.3.3 構造体型の定義

構造体は、メンバーと呼ばれる変数の集合体の型である。プログラマーが新たに型を定義するようなものである。定義の方法は、一般的には次のようになる。

```
struct タグ名 {  
    型 メンバー 1 名;  
    型 メンバー 2 名;  
    型 メンバー 3 名;  
    .  
    .  
    .  
    型 メンバー N 名;  
} 変数リスト;
```

例として学生の名前と成績、身長、体重を示した構造体を定義してみる。

```
struct gakusei{  
    char name[80];  
    int mathematics;  
    int english;  
    int japanese;  
    int electrical_eng;  
    int info_eng;  
    double height;  
    double weight;  
} sato, tanaka, yamamoto;
```

これで、gakusei 型の構造体変数の sato と tanaka、yamamoto が使えるようになる。さらに、watanabe という変数を追加したい場合は、

```
struct gakusei watanabe;
```

とすればよい。

1.3.4 構造体型のメンバーの参照

構造体のメンバーの参照には、. 演算子 (ドット演算子) をつかう。次のようにするのである。

| | |
|-------------------------------|--------------------|
| 構造体変数. メンバー | /* 通常 */ |
| 構造体変数. メンバー. メンバー | /* メンバーが構造体 */ |
| 構造体変数. メンバー. メンバー. メンバー | /* メンバーのメンバーも構造体*/ |
| 構造体変数 [配列の添え字]. メンバー | /* 構造体が配列*/ |
| 構造体変数. メンバー [配列の添え字] | /* メンバーが配列*/ |
| 構造体変数 [配列の添え字]. メンバー [配列の添え字] | /*構造体もメンバーも配列*/ |

1.3.5 プログラム例

構造体を用いたプログラム例としてリスト 1 を示す。この内容をよく理解する必要がある。

リスト 1: 構造体を使ったプログラム例

```
1 #include <stdio.h>
2 #include <string.h>
3
4 struct seiseki{
5     int mathematics;
6     int english;
7 };
8
9 struct gakusei{
10     char name[80];
11     struct seiseki test;
12     double height;
13 };
14
15 int main(void){
16
17     struct gakusei e2[10];
18
19     strcpy(e2[0].name,"yamamoto");
20     e2[0].test.mathematics = 95;
21     e2[0].test.english = 65;
22     e2[0].height = 174.8;
23
24     printf("%s\n", e2[0].name);
25     printf("  Mathematics : %d\n", e2[0].test.mathematics);
26     printf("    English      : %d\n", e2[0].test.english);
27     printf("    Height       : %f [cm]\n", e2[0].height);
28
29     return 0;
30 }
```

実行結果

```
yamamoto
  Mathematics : 95
    English    : 65
    Height     : 174.800000 [cm]
```

2 2進数と16進数

C言語の特徴であるポインタを学習するためには、2進数や16進数を理解する必要がある。そのために、10進数とそれらの関係について学習した。以下、その内容をまとめる。

- いろいろな数の表記方法がある。N進数の場合、次のようにN個の底で数を表現する。

2進数 0, 1

10進数 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

16進数 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- 桁上がりは、2進数の場合1の次で10に、10進数の場合9の次で10に、16進数の場合Fの次で10になる。
- 我々が通常用いている数の表現の意味は、次の通りである。数字の並び順序が重要で、これを「位取り記数法」と言う。

$$(1905)_{10} = (1 \times 10^3 + 9 \times 10^2 + 0 \times 10^1 + 5 \times 10^0)_{10}$$

- 基数の変換 (2 → 10進数)。通常の位取り記数法が理解できれば、簡単である。

$$\begin{aligned}(1101)_2 &= (1 \times 10^{11} + 1 \times 10^{10} + 0 \times 10^1 + 1 \times 10^0)_2 \\&= (1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)_{10} \quad \leftarrow \text{普通はここから計算} \\&= (8 + 4 + 0 + 1)_{10} \quad \leftarrow \text{ここから計算しても良い} \\&= (13)_{10}\end{aligned}$$

- 2進数の各桁の10進数の値(重み)を覚えておくと便利である。

$$1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096$$

- 基数の変換 (10→2進数)。2で割った余りを並べればよい。変換方法の例を、以下に示す。
 $(19)_{10} = (10011)_2$, $(2003)_{10} = (11111010011)_2$ である。

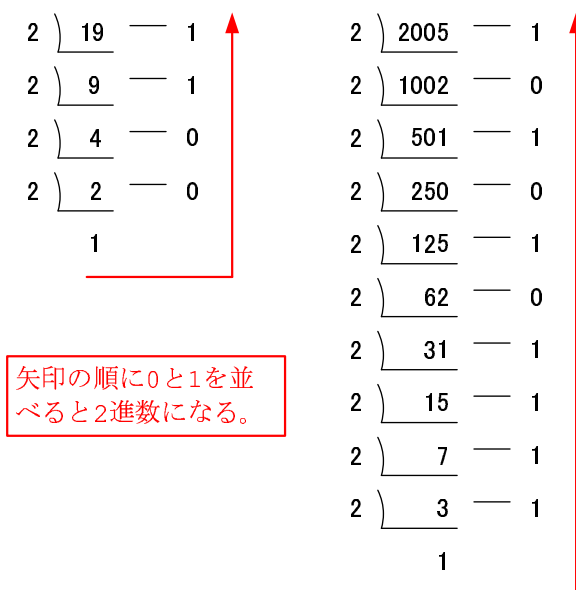


図 4: 10 進数から 2 進数への変換方法。

- 基数の変換 (16→10 進数)。これも、2 進数と同じ。

$$\begin{aligned}
 (376)_{16} &= (3 \times 10^2 + 7 \times 10^1 + 6 \times 10^0)_{16} \\
 &= (3 \times 16^2 + 7 \times 16^1 + 6 \times 16^0)_{10} \\
 &= (3 \times 256 + 7 \times 16 + 6 \times 1)_{10} \\
 &= (886)_{10}
 \end{aligned}$$

- 基数の変換 (10→16 進数)。16 で割って、その余りが各桁になる。

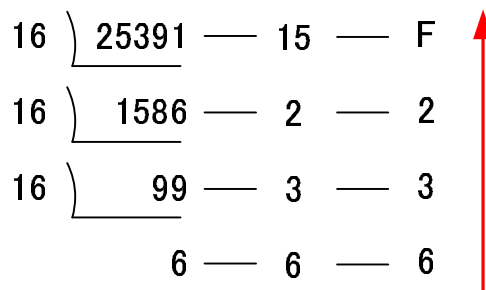


図 5: 10 進数から 16 進数への変換方法。

- 基数の変換 (2↔16 進数)。2 進数の 4 桁が、16 進数の 1 桁に等しいことを利用する。

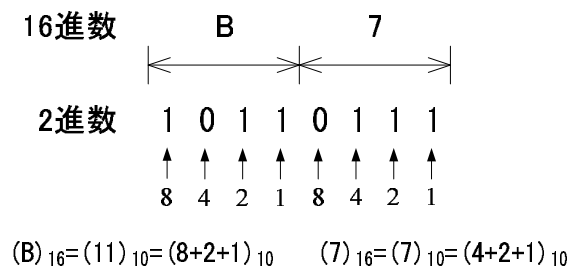


図 6: 2 進数と 16 進数の相互変換

- 桁数が合わない場合は、先頭に必要なだけゼロを書き足して考える。例えば、 $(101100)_2 = (00101100)_2 = (2C)_{16}$ となる。

3 ポインター

3.1 コンピューターの構造

ポインターは、メモリと密接に関わっている。ポインターを学習する前に、簡単にコンピューターの構造を学習した。その内容は、以下の通り。

- 図7がコンピューターの基本構成である。
- 制御装置と演算装置、記憶装置、入力装置、出力装置をコンピューターの五大装置と呼び、それぞれには以下の働きがある。

制御装置 主記憶装置（メインメモリ）に格納されているプログラムを受け取り、各装置に指令を出す。

演算装置 主記憶装置の中にあるデータを受け取り、制御装置の指令に従い、それを処理する。

記憶装置 主記憶装置と補助記憶装置がある。

主記憶装置 命令とデータからなるプログラムを格納する。

補助記憶装置 大容量、あるいは半永久的に残したいデータを補助記憶装置に格納する。

入力装置 コンピューターの外部からデータを取り込む装置である。

出力装置 主記憶装置に格納されているデータをコンピューター外部に出力する装置である。

- 制御装置と演算装置をまとめて中央制御装置 (CPU:Central Processing Unit) あるいは、MPU(Micro Processing Unit) と言う。

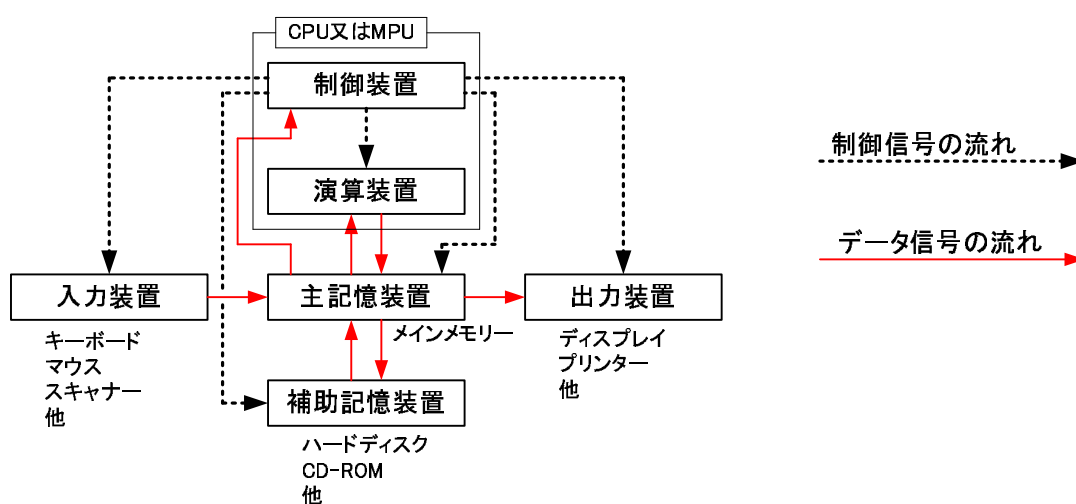


図7: コンピューターの基本構成

3.2 メモリーとデータ

- 2進数の1桁が1ビットである。
- 諸君が使っているパソコンのアドレスは32ビットで表現されている。これは16進数では8桁である。
- 一つのアドレスに8ビット(1バイト)記憶できる。これは、16進数2桁で表現できる。
- 8ビットを1バイトと言う。
- 諸君が使っているコンパイラでは、変数は表1に示しているバイト数で表現されている。

表 1: 変数の型とバイト数

| 型名 | データ型 | バイト数 | ビット数 |
|-------|--------|------|------|
| 文字型 | char | 1 | 8 |
| 整数型 | int | 4 | 32 |
| 倍精度実数 | double | 8 | 64 |

- プログラムは命令とデータから構成され、いずれもメモリーの中に格納される。
- プログラムの関数(これが命令)が格納されるアドレスは、関数名で参照できる。
- データが格納されるアドレスは、データ名の前に&を付けることで参照できる。&はアドレス演算子である。
- アドレスの表示には変換指定子%pを使う。
- ローカル変数は名前が同じでも、メモリーの配置場所は異なる。正確言うと、その関数が呼び出されたときのみ、ローカル変数はメモリーに割り当てられる。

3.3 ポインター

- ポインターとは、アドレスを格納する変数のことである。
- ポインターの宣言には、型名とアスタリスク(*)を付ける。

```
int *pi;  
double *px;
```

- 変数のアドレスを取り出すには、変数名の前にアンパサンド(&)をつける。&はアドレス演算子である。以下の例では、piとpxがポインターで、変数iとxのアドレスを、それに代入している。

```
pi=&i;  
px=&x;
```

- ポインタが示しているデータの値を取り出すためには、ポインタ変数の前にアスタリスク(*)をつける。*は間接参照演算子である。つぎの例は、ポインタ pi と px が指し示しているアドレスにある値を取り出して、変数 j と y に代入している。

```
j=*pi;
y=*px;
```

- リスト 2 のプログラムをよく理解すること。

4 行 整数型のポインタ p を宣言している。p に整数型のデータの先頭アドレスを格納する。

5 行 整数型の変数 i を宣言し、 $(11223344)_{16}$ を代入している。

7 行 変数 i の先頭アドレスをアドレス演算子 &により取り出し、ポインタ p に代入している。

9 行 整数変数 i の先頭アドレスを変換指定子 %p により表示している。

10 行 ポインタ p の先頭アドレスを変換指定子 %p により表示している。

12 行 整数変数 i の値を 16 進数表示の変換指定子 %0x により表示している。

13 行 ポインタ p の値を 16 進数表示の変換指定子 %0x により表示している。ただし、ポインタはアドレスなので、強制型変換 (キャスト) により、符号なし整数にしている。

15 行 ポインタが指し示すアドレスに格納されているデータを表示している。

リスト 2: ポインタとアドレス

```
1 #include <stdio.h>
2
3 int main(void){
4     int *p;
5     int i=0x11223344;
6
7     p=&i;
8
9     printf("address i %p\n", &i);
10    printf("address p %p\n", &p);
11
12    printf("value i %0x\n", i);
13    printf("value p %0x\n", (unsigned int)p);
14
15    printf("value *p %0x\n", *p);
16
17    return 0;
18 }
```

実行結果

```
address i 0xbffff6b0
address p 0xbffff6b4
value i 11223344
value p bffff6b0
value *p 11223344
```

この実行結果から、メモリーは図 8 のようになっていることが分かる。

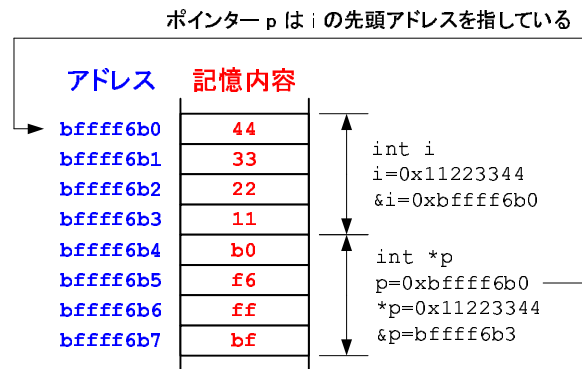


図 8: リスト 2 のプログラム実行後のメモリーの内容