



ce-7. メモリ内でのデータ配置

(C プログラミング応用) (全 1 4 回)

URL: <https://www.kkaneko.jp/pro/c/index.html>

金子邦彦



例題 1 . 棒グラフを描く



- 整数の配列から, その棒グラフを表示する
 - ループの入れ子で, 棒グラフの表示を行う (参考: 第 6 回授業の例題 3)
- 棒グラフの 1 本の棒を画面に表示する機能を持った関数を補助関数として作る



例題：棒グラフ

```
#include "stdio.h"
#include <math.h>
void draw_bar( int len )
{
    int i;
    for (i=0; i<len; i++) {
        printf("*");
    }
    printf("¥n");
    return;
}
```

補助関数

draw_bar

棒グラフの1本の棒を
画面に表示する機能

```
int main()
{
    int i;
    int ch;
    int a[]={6,4,7,1,5,3,2};
    for (i=0; i<7; i++) {
        draw_bar(a[i]);
    }
    ch = getchar();
    ch = getchar();
    return 0;
}
```

配列の宣言

配列から読み出して、補助関数 draw_bar に渡している

各自でビルド、実行して、
動作を確認して下さい •3

実行結果の例



```
*****  
****  
*****  
*  
*****  
***  
**
```

棒グラフが表示される

プログラム実行順



- 普通, プログラム中の文は, 上から下へ順に実行される
- 関数呼び出しでは, 関数の先頭に「ジャンプ」する.

このことは, C言語が「手続き型言語」と言われる理由の1つ

関数呼び出しの例) `draw_bar(a[i]);`

- 呼び出された関数の中で `return` 文に出会うと, 関数呼び出しの場所に戻る.



プログラム実行順

```
#include "stdio.h"
#include <math.h>
void draw_bar( int len )
{
    int i;
    for (i=0; i<len; i++) {
        printf("*");
    }
    printf("¥n");
    return;
}
```

関数呼び出しの場所に戻る

複数の関数を含む
プログラム

補助関数

プログラム実行は
メインの関数から
始まる

```
int main()
{
    int i;
    int ch;
    int a[]={6,4,7,1,5,3,2};
    for (i=0; i<7; i++) {
        draw_bar(a[i]);
    }
    ch = getchar();
    ch = getchar();
    return 0;
}
```

7回の繰り返し

関数 draw_bar を呼び出し

メインの関数



メインの関数内の変数

```
#include "stdio.h"
#include <math.h>
void draw_bar( int len )
{
    int i;
    for (i=0; i<len; i++) {
        printf("*");
    }
    printf("¥n");
    return;
}
```

```
int main()
{
    int i;
    int ch;
    int a[]={6,4,7,1,5,3,2};
    for (i=0; i<7; i++) {
        draw_bar(a[i]);
    }
    ch = getchar();
    ch = getchar();
    return 0;
}
```

変数 i, ch, a
をメモリエリア中に確保

ここで使用



メインの関数内の変数

```
#include "stdio.h"
#include <math.h>
void draw_bar( int len )
{
    int i;
    for (i=0; i<len; i++) {
        printf("*");
    }
    printf("¥n");
    return;
}
```

```
int main()
```

```
{
    int i;
    int ch;
    int a[]={6,4,7,1,5,3,2};
    for (i=0; i<7; i++) {
        draw_bar(a[i]);
    }
    ch = getchar();
    ch = getchar();
    return 0;
}
```

この時点では

変数名	値	タイプ
i	7	int
ch	未使用	int
a	6,4,7,1,5,3,2	int の配列





実際のメモリの中身 **a**

16 進数でメモリの中身を表示

```

: 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f | 0123456789abcdef
-----
0012FEA0 : cc cc cc cc 6 0 0 0 4 0 0 0 7 0 0 0 | .....
0012FEB0 : 1 0 0 0 5 0 0 0 3 0 0 0 2 0 0 0 | .....
0012FEC0 : cc cc cc cc cc cc cc cc cc cc cc cc cc cc | .....
0012FED0 : cc cc cc cc 7 0 0 0 cc cc cc cc c0 ff 12 0 | .....
0012FEE0 : 10 26 41 0 1 0 0 0 0 14 37 0 a8 14 37 0 | .&A.....7...7.
0012FEF0 : 94 0 0 0 5 0 0 0 1 0 0 0 28 a 0 0 | .....(
0012FF00 : 2 0 0 0 53 65 72 76 69 63 65 20 50 61 63 6b | ....Service Pack
0012FF10 : 20 32 0 0 1 0 0 0 0 3 0 0 0 0 0 0 | 2.....
0012FF20 : 0 0 0 0 1a 0 0 1 4c 7c 8b f6 d4 f 0 0 | .....L|.....

```

i

ch

変数名	値	タイプ
i	7	int
ch	未使用	int
a	6,4,7,1,5,3, 2	int の配列



補助関数内の変数

```
#include "stdio.h"
#include <math.h>
void draw_bar( int len )
{
  int i;
  for (i=0; i<len; i++) {
    printf("*");
  }
  printf("¥n");
  return;
}
```

変数 **len, i**
がメモリエリア中に確保される

ここで使用

```
int main()
{
  int i;
  int ch;
  int a[]={6,4,7,1,5,3,2};
  for (i=0; i<7; i++) {
    draw_bar(a[i]);
  }
  ch = getchar();
  ch = getchar();
  return 0;
}
```

変数 **i, ch, a**
がメモリエリア中に確保される

ここで使用



補助関数内の変数

```
#include "stdio.h"
#include <math.h>
void draw_bar( int len )
{
    int i;
    for (i=0; i<len; i++) {
        printf("*");
    }
    printf("¥n");
    return;
}
```

```
int main()
{
    int i;
    int ch;
    int a[]={6,4,7,1,5,3,2};
    for (i=0; i<7; i++) {
        draw_bar(a[i]);
    }
    ch = getchar();
    ch = getchar();
    return 0;
}
```

この時点では

変数名	値	タイプ
len	6 など	int
i	6 など	int

len は, 6, 4, 7, 1, 5, 3, 2 の値を取り,
i の値は len と等しい

実際のメモリの中身



変数名	値	タイプ
len	6 など	int
i	6 など	int

len i

16 進数でメモリの中身を表示

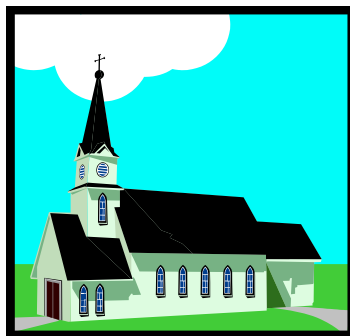
	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	01	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0012FDB0	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	6	0	0	0
0012FDC0	cc	cc	cc	cc	dc	fe	12	0	44	1d	41	0	6	0	0	0
0012FDD0	4f	5b	95	7c	40	0	0	0	0	f0	fd	7f	cc	cc	cc	cc	0	[.	@
0012FDE0	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0012FDF0	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0012FE00	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0012FE10	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0012FE20	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0012FE30	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc

ページ 10 で見たメモリの中身とは、メモリアドレスが違う
(個々の関数ごとに、個別のメモリエリアが割り当てられる)

家と住所



Aさんの家



福岡市東区
箱崎1丁目1番

Bさんの家



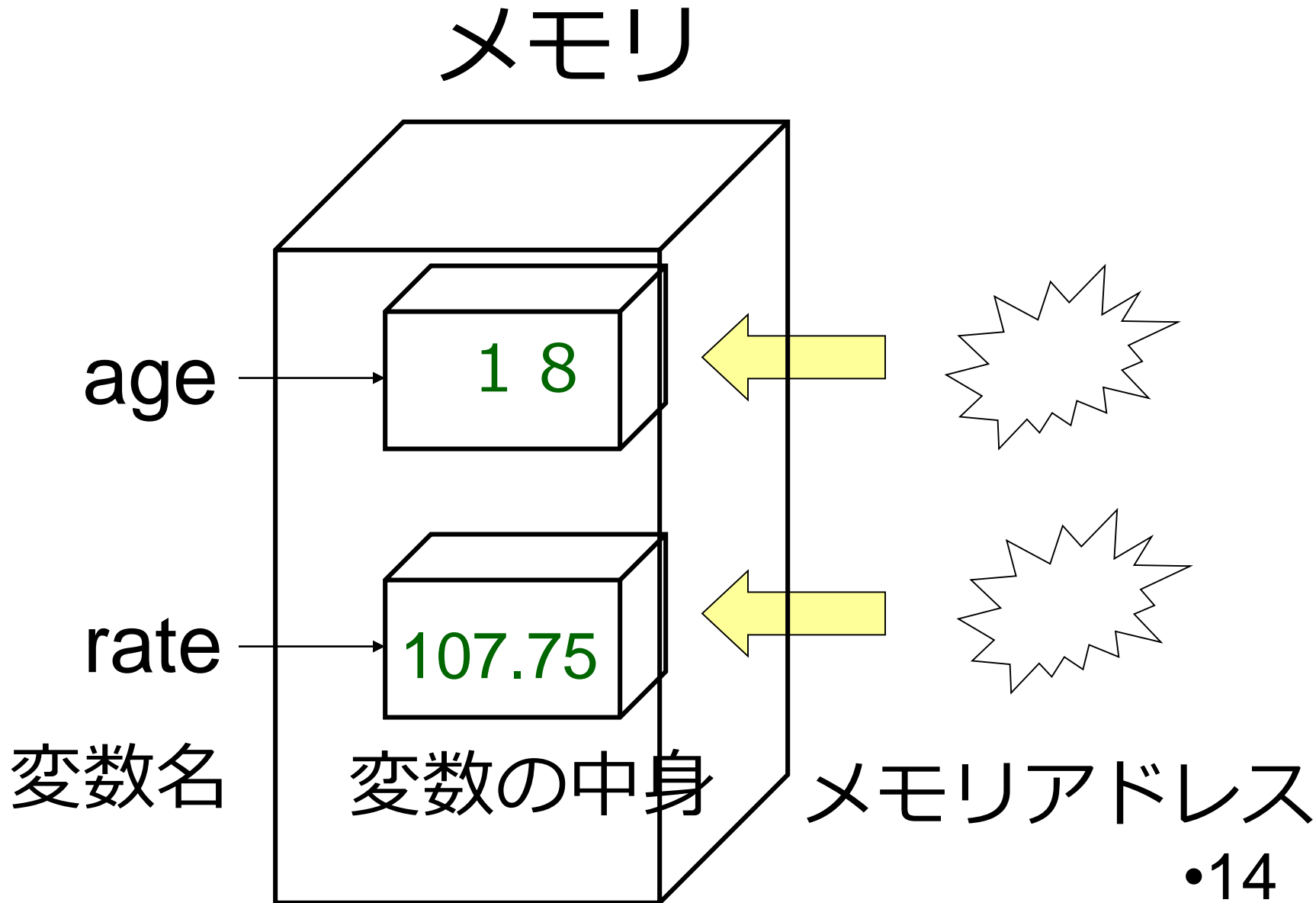
福岡市東区
箱崎2丁目2番

名前

家

住所

メモリアドレスとは



メモリアドレスとは



- すべてのデータには「メモリアドレス」が付けられている

変数の中身： 値

「18」 「107.75」 など

変数名： プログラム内で使うための名前

「age」, 「rate」 など

メモリアドレス：

変数のそれぞれに付けられた「住所」の
ようなもの

例題 2. 変数のメモリアドレス表示



- 次の3つの変数を使って、「底辺と高さから、3角形の面積を計算するプログラム」を作る。

底辺 teihen 浮動小数データ

高さ takasa 浮動小数データ

面積 menseki 浮動小数データ

- これら変数のメモリアドレスの表示も行う



```
#include "stdio.h"
#include <math.h>
int main()
{
    double teihen;
    double takasa;
    double menseki;
    int ch;
    teihen = 3;
    takasa = 4;
    menseki = teihen * takasa * 0.5;
    printf("menseki = %f¥n",menseki);
    printf("address(teihen) = %p¥n", &teihen );
    printf("address(takasa) = %p¥n", &takasa );
    printf("address(menseki) = %p¥n", &menseki );
    ch = getchar();
    ch = getchar();
    return 0;
}
```

「&」はメモリアドレス
の取得

「%p」はメモリアドレス
の表示

変数のメモリアドレス表示



実行結果の例

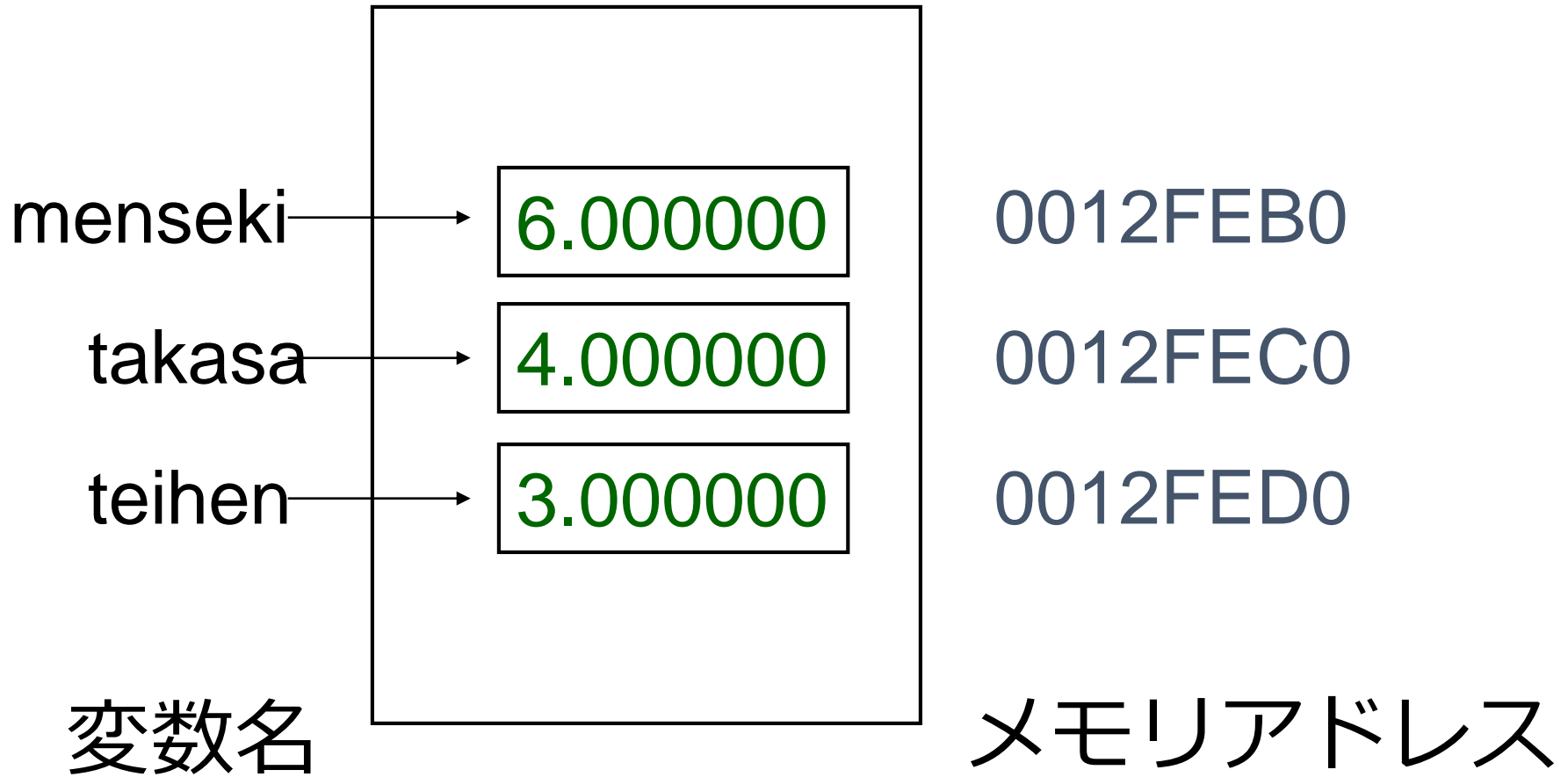
```
menseki = 6.000000  
address(teihen) = 0012FED0  
address(takasa) = 0012FEC0  
address(menseki) = 0012FEB0
```

表示された
メモリアドレス*

メモリアドレスの値が
ここでの「例」と違っている
ことはある（動作は正しい）



メモリ（模式図）



実際のメモリの中身

menseki
takasa



```

menseki = 6.000000
address(teihen) = 0012FED0
address(takasa) = 0012FEC0
address(menseki) = 0012FEB0
16 進数でメモリの中身を表示
      : 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f | 0123456789abcdef
-----|-----
0012FEB0 : 0 0 0 0 0 0 18 40 cc cc cc cc cc cc cc cc | .....@.....
0012FEC0 : 0 0 0 0 0 0 10 40 cc cc cc cc cc cc cc cc | .....@.....
0012FED0 : 0 0 0 0 0 0 8 40 cc cc cc cc c0 ff 12 0 | .....@.....
0012FEE0 : 10 28 41 0 1 0 0 0 0 14 37 0 a8 14 37 0 | .(A.....7...7.
0012FEF0 : 94 0 0 0 5 0 0 0 1 0 0 0 28 a 0 0 | .....(...
0012FF00 : 2 0 0 0 53 65 72 76 69 63 65 20 50 61 63 6b | ....Service Pack
0012FF10 : 20 32 0 0 2 0 0 0 0 3 0 0 0 0 0 0 | 2.....
0012FF20 : 0 0 0 0 10 0 0 1 4c fc c6 ad c6 23 0 0 | .....L....#..
    
```

teihen

double 型の変数は
8 バイトになっている

変数名	値	タイプ
menseki	6	double
takasa	4	double
teihen	3	double

メモリアドレスの取得と表示



```
printf("address(teihen) = %p¥n", &teihen );
```

メモリアドレスの表示 メモリアドレスの取得

- 変数からメモリアドレスの取得
 - &: メモリアドレスを取得するための演算子
変数名（など）の前に付ける
- メモリアドレスの表示のための書式
 - %p: メモリアドレスを表示せよという指示
printf 文などで使用

例題 2 b . 配列のメモリアドレス



- 次の2つの配列を使って, ベクトル (1.9, 2.8, 3.7) と, ベクトル (4.6, 5.5, 6.4) の内積を求めるプログラムを作る.

ベクトル (1.9, 2.8, 3.7) u 要素数3の浮動小数の配列

ベクトル (4.6, 5.5, 6.4) v 要素数3の浮動小数の配列

- これら配列の要素について, メモリアドレスの表示も行う



各自でビルド,
実行して下さい

```
#include "stdio.h"
#include <math.h>
int main()
{
    double u[]={1.9, 2.8, 3.7};
    double v[]={4.6, 5.5, 6.4};
    int i;
    double ip;
    int ch;
    ip = 0;
    for (i=0; i<3; i++) {
        ip = ip + u[i]*v[i];
    }
    printf("内積=%f\n", ip);
    printf("address(u[0]) = %p\n", &u[0]);
    printf("address(u[1]) = %p\n", &u[1]);
    printf("address(u[2]) = %p\n", &u[2]);
    printf("address(v[0]) = %p\n", &v[0]);
    printf("address(v[1]) = %p\n", &v[1]);
    printf("address(v[2]) = %p\n", &v[2]);
    ch = getchar();
    ch = getchar();
    return 0;
}
```

「{1.9, 2.8, 3.7}」のように書いて,
u[0], u[1], u[2] に値をセット

「&」はメモリアドレス
の取得

「%p」はメモリアドレス
の表示

配列のメモリアドレス



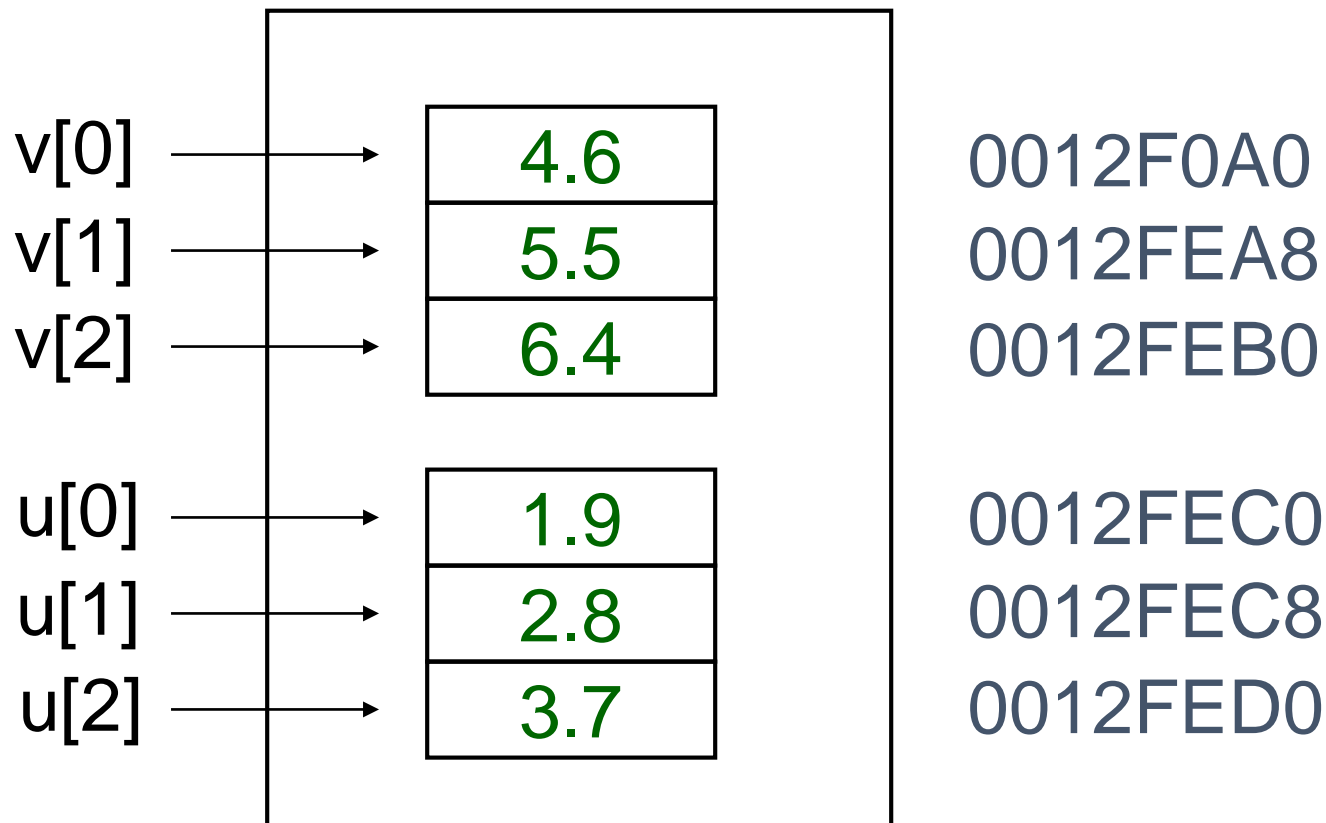
実行結果の例

表示された
メモリアドレス

```
内積=47.820000  
address(u[0]) = 0012FEC0  
address(u[1]) = 0012FEC8  
address(u[2]) = 0012FED0  
address(v[0]) = 0012FEA0  
address(v[1]) = 0012FEA8  
address(v[2]) = 0012FEB0
```




メモリ (模式図)

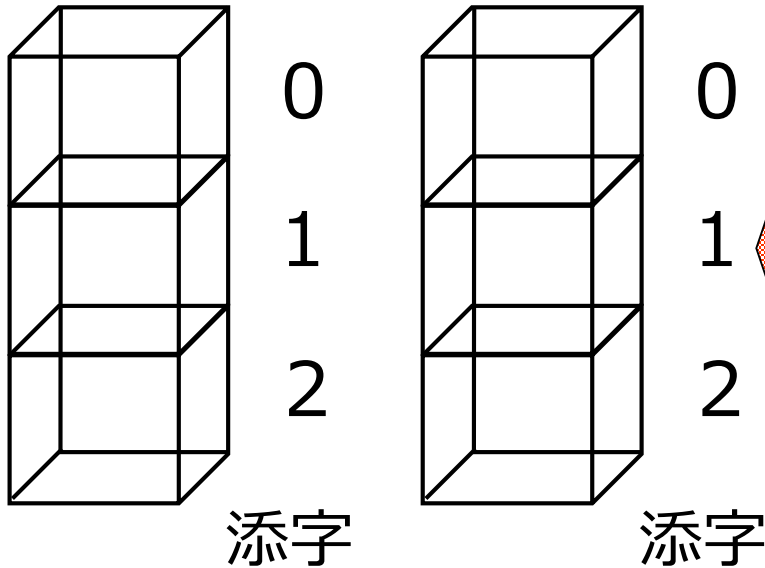


メモリアドレス

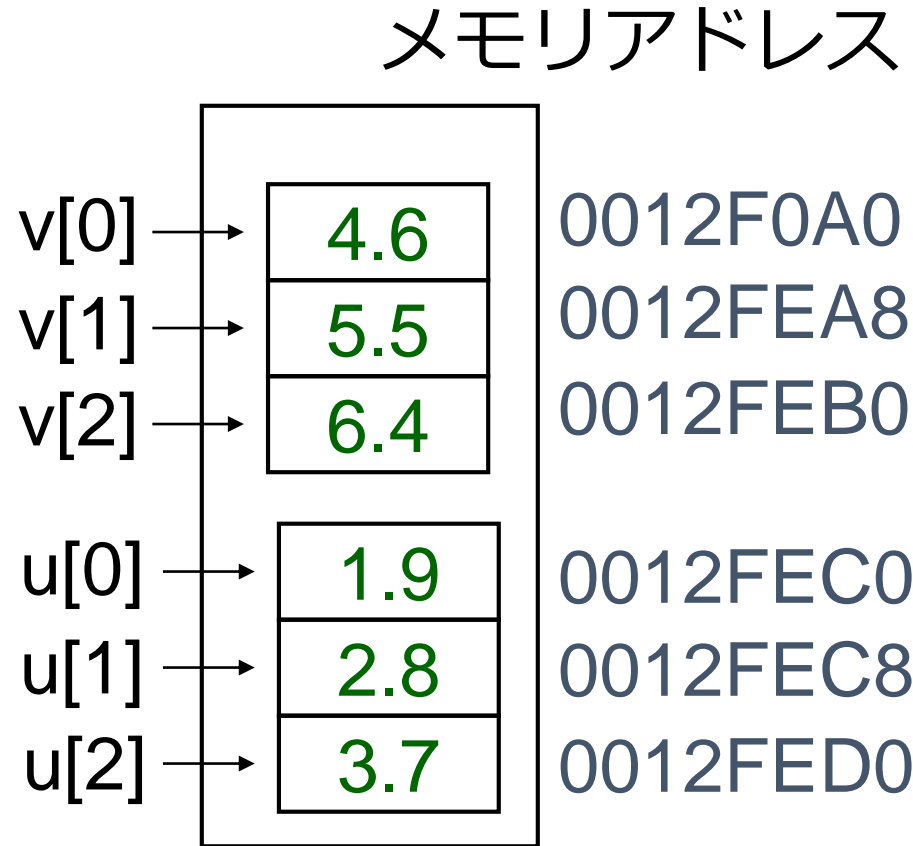
配列とメモリアドレス



配列 u (サイズは3) 配列 v (サイズは3)



2つの配列



メモリ内の配置
(配列の並びはそのままで
メモリに入る)

実際のメモリの中身



v[0],v[1],v[2]

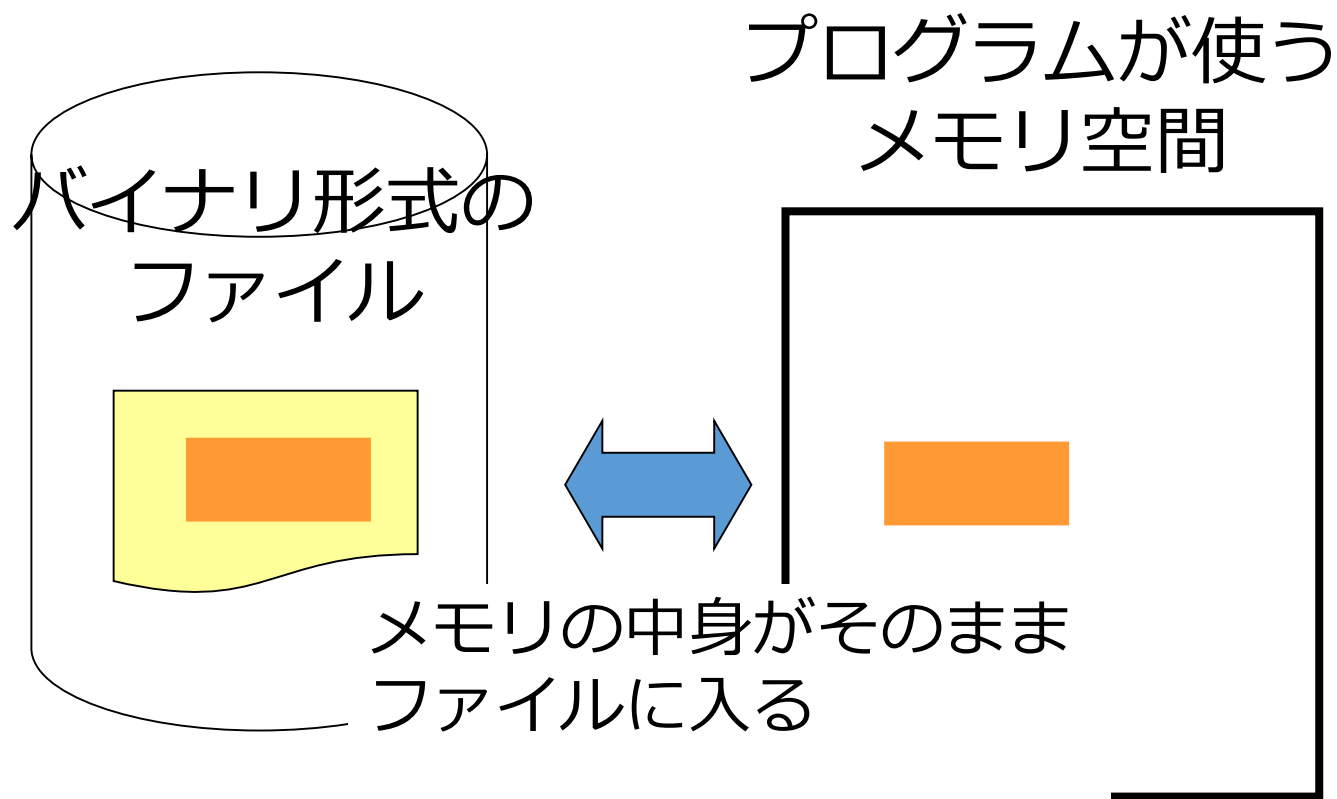
16 進数でメモリの中身を表示

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	0123456789abcdef
0012FEA0	66	66	66	66	66	66	12	40	0	0	0	0	0	0	16	40	ffffff.@.....@
0012FEB0	9a	99	99	99	99	99	19	40	cc	cc	cc	cc	cc	cc	cc	cc@.....
0012FEC0	66	66	66	66	66	66	fe	3f	66	66	66	66	66	66	6	40	ffffff.?ffffff.@
0012FED0	9a	99	99	99	99	99	d	40	cc	cc	cc	cc	c0	ff	12	0@.....
0012FEE0	90	28	41	0	1	0	0	0	0	14	37	0	a8	14	37	0	.(A.....7...7.
0012FEF0	94	0	0	0	5	0	0	0	1	0	0	0	28	a	0	0(...
0012FF00	2	0	0	0	53	65	72	76	69	63	65	20	50	61	63	6bService Pack
0012FF10	20	32	0	0	2	0	0	0	0	3	0	0	0	0	0	0	2.....

u[0],u[1],u[2]

double 型の変数は
8 バイトになっている

バイナリ形式のファイル



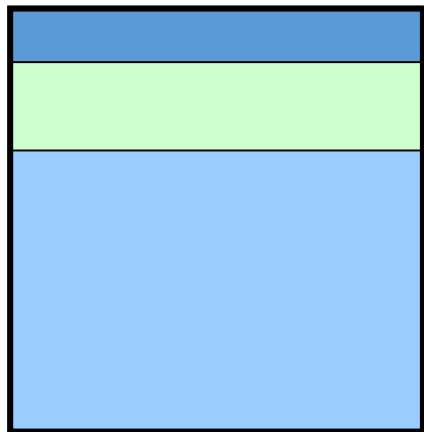
読み出し : fread を使用
書き込み : fwrite を使用

Windows ビットマップファイル



Windows ビットマップファイルの例

786,484 バイトのファイル
(バイナリ形式)



先頭 14 バイト : ファイルヘッダ

先頭 40 バイト* : ヘッダ

(画像の種類によっては40より長い)

残り : 画像データの本体

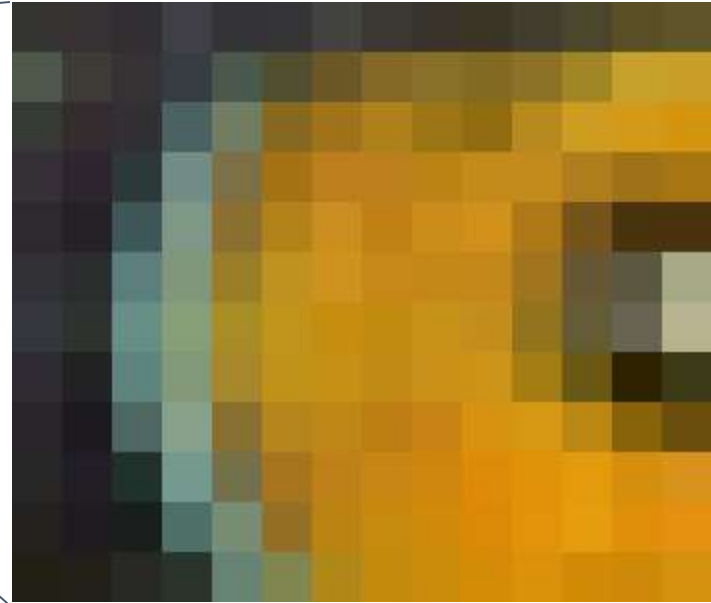
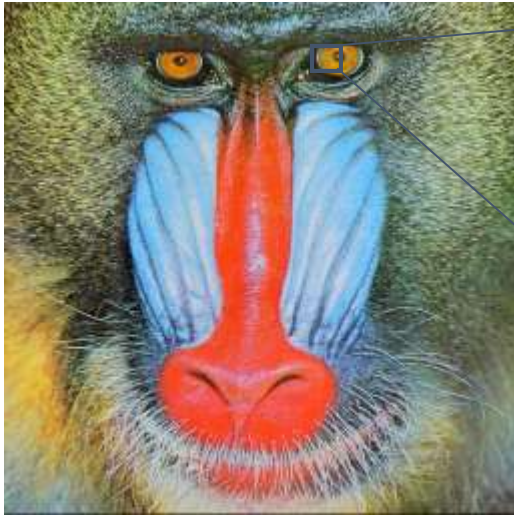
画像データの本体が
先頭から何バイト目か
(bfOffBits) など

幅 (biWidth)

高さ (biHeight)

1画素あたりのビット数
(biBitCount)

圧縮法 (BiCompression)
→ 0 なら圧縮していない など



それぞれの格子が画素



参考： メモリダンプ用プログラム

```
#include "stdio.h"
```

```
void dump_line( unsigned char *top_address )
```

```
{
```

```
    unsigned char *p;
```

```
    // 「%p」はポインタの表示
```

```
    printf( "%p :", top_address );
```

```
    // 以下, 16進数での表示を16バイト分行う
```

```
    for ( p = top_address; p < (top_address + 16); p++ ) {
```

```
        // 「%2x」は16進数2桁での表示
```

```
        printf( " %2x", *p );
```

```
    }
```

```
    // メモリの中身が, 16進数で 20 以上 7e 以下のときは, 「文字」もで表示
```

```
    printf( "|" );
```

```
    for ( p = top_address; p < (top_address + 16); p++ ) {
```

```
        // 「%c」は文字 ( 1 文字 ) での表示
```

```
        if ( ( *p ) >= 0x20 ) && ( *p ) < 0x7e ) {
```

```
            printf( "%c", *p );
```

```
        }
```

```
        else {
```

```
            printf( "." );
```

```
        }
```

```
    }
```

```
    printf( "¥n" );
```

```
}
```



// 16進数でメモリの中身を表示

```
void dump( unsigned char *address, int len )
```

```
{  
    unsigned char *current;  
    printf( "16 進数でメモリの中身を表示¥n" );  
    printf( "      : 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f|0123456789abcdef¥n" );  
    printf( "-----:-----|-----¥n" );  
    current = (unsigned char*) ( ( (int) address ) / 16 ) * 16 );  
    while ( current < (address + len) ) {  
        dump_line( current );  
        current = current + 16;  
    }  
    return;  
}
```

```
}  
int main()  
{  
    char s[] = "89771843";  
    int ch;  
    dump( (unsigned char*)s, 16 );  
    printf( "Enter キーを1,2回押すと、プログラムが終了します¥n" );  
    ch = getchar();  
    ch = getchar();  
    return 0;  
}
```

