

プログラミング発展

2023年度2Q 火曜日5~7時限(13:45~16:30)
金曜日5~7時限(13:45~16:30)

工学院 情報通信系

尾形わかは, 松本隆太郎,
Chu Van Thiem, Saetia Supat
TA:東海林郷志, 千脇彰悟
(最終更新 : 6月14日15 : 00)

変数とメモリ 1

- 変数や配列を定義すると、プログラム実行時に、その変数の値を保存する領域（メモリ）を確保して、変数に割り当てる。
- 変数の種類によって、メモリが確保されるタイミングや解放（確保をやめる）するタイミングが異なる。
- 当然、メモリは有限。
 - トラブルが起こる可能性

文字型変数の表現

```
char str[6]={'H', 'e', 'l', 'l', 'o', '¥0'}
```

メモリのアドレス	メモリの中身	配列と要素
addr	H	str[0]
addr+1	e	str[1]
addr+2	l	str[2]
addr+3	l	str[3]
addr+4	o	str[4]
addr+5	¥0	str[5]

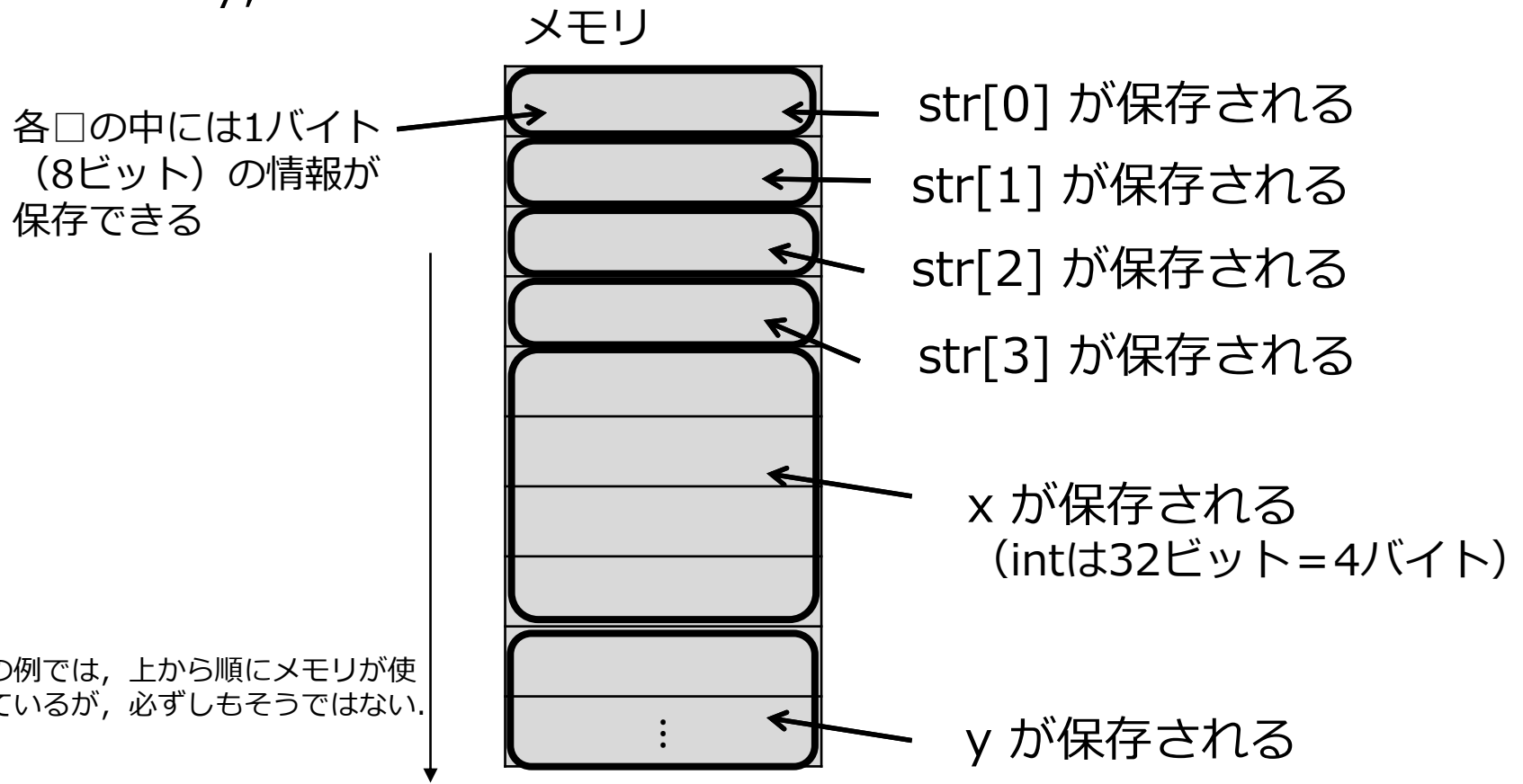
変数とメモリ 2

- 例

```
char str[4];
```

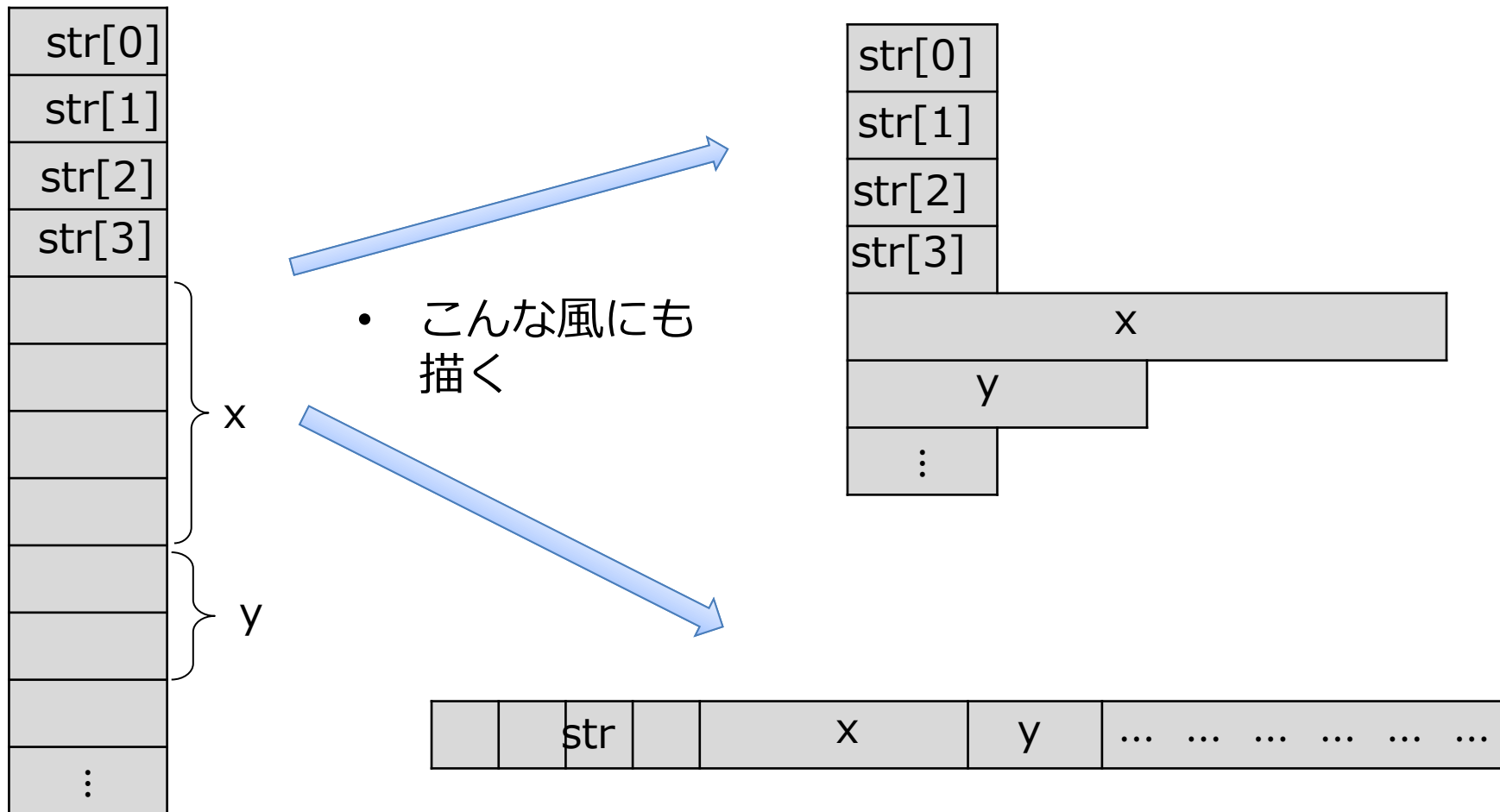
```
int x;
```

```
short y;
```



変数とメモリ 3

- メモリは、同じサイズのものが一元的に並んでいるが



大域変数と局所変数

- 関数の外側で宣言した変数（や配列）を大域（グローバル）変数（配列）と呼び、どこからでも値を読み書きできる
- 関数の内側で宣言した変数（や配列）を局所（ローカル）変数（配列）と呼び、宣言した関数の中からその名前で値を読み書きできる

二種類の局所変数

- 局所変数を宣言するときに、型の前に「auto」または「static」を付けることができる。省略すると「auto」が付けられたと見なされる。これらを自動変数ならびに静的変数と呼ぶ

- 例:

```
auto int number1;
```

```
static int number2;
```

自動変数と静的変数の違い

```
#include <stdio.h>
#include <time.h>
int counter; // 大域変数は初期化しないとゼロに初期化される
int main(void) {
    static clock_t static_time; ; // 初期値は0
    auto clock_t auto_time /* 初期値未定 */ ; volatile int i=0;
    static_time = auto_time = clock();
    while (i <= 10000000) ++i; // 時間つぶし
    if (counter < 3) { ++counter; main(); }
    printf("static=%ld at %p, auto=%ld at %p¥n",
        static_time,&static_time,auto_time,&auto_time);
    return 0;
}
```

上記プログラムを実行すると...

counter = 0

main

auto_time = t1

static_time = t1

counter = 1

再帰呼び出し

main

auto_time = t2

static_time = t2

counter = 2

再帰呼び出し

main

auto_time = t3

static_time = t3

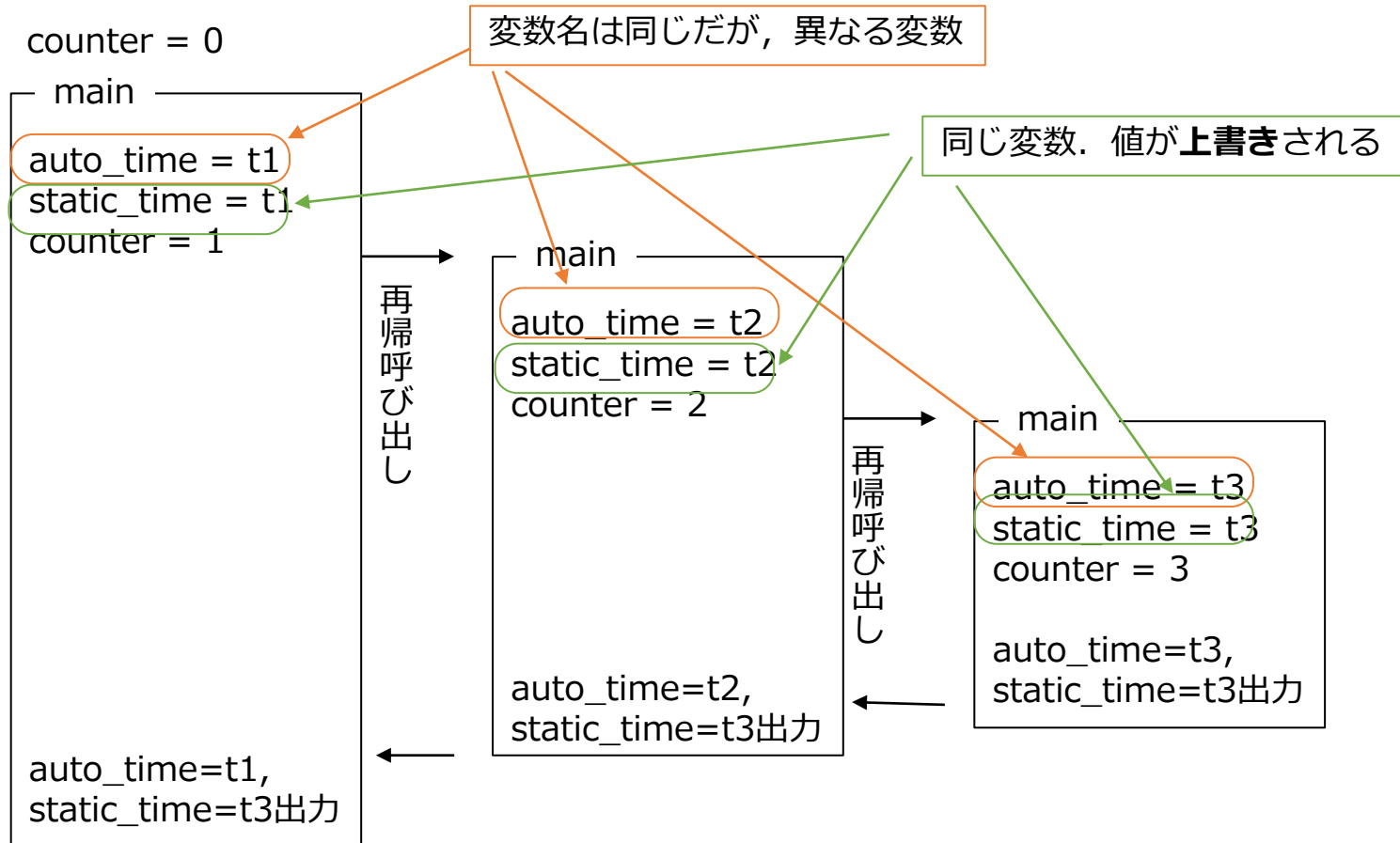
counter = 3

auto_time=t3,
static_time=t3出力

auto_time=t1,
static_time=t3出力

変数名は同じだが、異なる変数

同じ変数. 値が**上書き**される



実行結果

```
static=49324 at 0x55c6325c7018, auto=49324 at 0x7ffd68ec70f0  
static=49324 at 0x55c6325c7018, auto=34933 at 0x7ffd68ec7120  
static=49324 at 0x55c6325c7018, auto=20482 at 0x7ffd68ec7150  
static=49324 at 0x55c6325c7018, auto=2721 at 0x7ffd68ec7180
```

関数main()を呼び出しその中で局所変数を書き換えたときに、静的変数は呼び出し側の変数の値が変わり、自動変数は変わらない。静的変数はたった一つだけあるが、自動変数は呼び出された関数ごとにある。この違いは再帰的呼び出しや、複数CPUコアからの並列化された呼び出しのときに違いを生み出す（2つのCPUコアが一つの静的変数を同時に1増やすと、タイミングによって1だけ増えたり2増えたりするなど）。

静的変数と自動変数の違い

- 自動変数は、その自動変数を宣言した関数を呼び出したときに新しく別のメモリをその変数に割り当てる。その関数からreturnするとその割り当てメモリは解放され他の変数に割り当てられる。自動配列の大きさは実行時に決めてよい(scanf等で)
- 静的変数は、宣言した関数を呼び出す前に割り当てられプログラム終了時まで同じメモリが割り当たったままである。静的配列の大きさはコンパイル時に定数として決められる必要がある。

局所変数へのメモリ割り当て

- メモリ量は有限だから局所変数にメモリ割り当てできないことがある
- 静的変数に割り当てるメモリが足りないときは、プログラム起動時にいきなり割り当て失敗して終了する（プログラムが起動しない）
- 自動変数は何回か関数を呼び出したあとに割り当て失敗してプログラムが異常終了する。
処理するデータが少ないときに正常動作し、多くなると異常動作する原因の一部はこれ

メモリ割り当て失敗の把握

- 局所変数の割り当てに失敗していることを把握することは困難であり、割り当て失敗時にどういう動作をするかは実行環境にかなり依存する
- 次のページに自動変数に割り当てられる最大メモリ量を調べるプログラムを掲載する。無限ループしているプログラムなので、**必ずメモリ割り当て失敗して異常終了する**

```
#include <stdio.h>
```

```
int main() {  
    const int KiB=1024;  
    int i;  
    for (i=1; ; i++) {  
        char buf[i * KiB];  
        buf[0] = 100;  
        buf[sizeof buf - 1] = 100;  
        printf("%d KiB can be used.¥n", i);  
        fflush(stdout);  
    }  
    return 0;  
} /* これを実行すると... */
```

自動変数割り当て失敗時の動作

```
1  #include <stdio.h>
2
3  int main() {
4      const int KiB=1024;
5      int i;
6      for (i=1; ; i++) {
7          char buf[i * KiB];
8          buf[0] = 100;
9          buf[sizeof buf - 1] = 100;
10         printf("%d KiB can be used.\n", i);
11         fflush(stdout);
12     }
13     return 0;
14 }
```

8166 KiB can be used.
8167 KiB can be used.
8168 KiB can be used.
8169 KiB can be used.
(lldb)

「構造体(1)」

1. 構造体
2. 共用体
3. 列挙型

構造体1 概念

例：住所録

情報（入力）：氏名，性別，住所

単純な配列で表現することも可能だが．．．．

```
#define N_PERSON 50    /* #person */
char pdata_name[N_PERSON][40];
int  pdata_gender [N_PERSON];
char pdata_address [N_PERSON][100];
```

pdata_name		Taro		
pdata_gender		1		
pdata_address				Ookayama

..... それぞれの配列は独立

→相互の関連はプログラマのみ把握

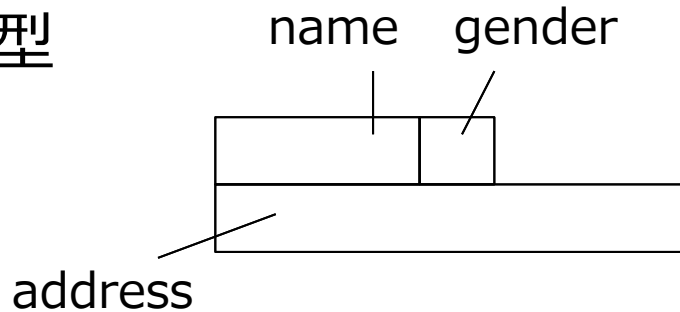
構造体1 概念（続き）

例：住所録

情報（入力）：氏名，性別，住所

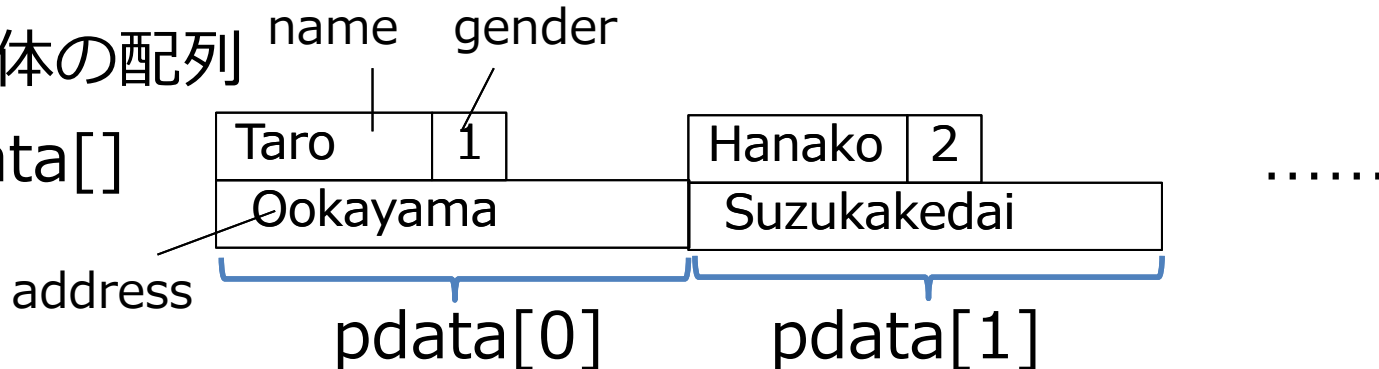
これらをひとまとめのデータとして取り扱えれば，
データ管理が分かりやすくなる → 構造体

構造体の
データ型



構造体の配列

pdata[]




構造体2

例：住所録

情報（入力）：氏名，性別，住所

```
#define N_PERSON 50 /* #person */
```

```
struct pdata_st {  
    char name[40];      /* Name */  
    int  gender;        /* Gender(M1, F2) */  
    char address[100];  /* Address */  
};
```



構造体の宣言

```
struct pdata_st pdata [N_PERSON]; /* 50個の構造体からなる配列を宣言している */
```

各エントリ("pdata")の情報をひとまとめに

→相互の関連が明確に

構造体3

例：住所録

情報（入力）：氏名，性別，住所

```
#define N_PERSON 50

struct pdata_st {
    char name[40];
    int  gender;
    char address[100];
};

struct pdata_st pdata[N_PERSON];
```

構造体の名称

構造体を構成する要素
*構造体を更に要素とすることもできる
（宣言している構造体を含む）

実際の変数（配列）

上記の構造体はEx 3-2で用いるが、そこでは男性のときにgenderを整数1に設定し、女性のときにgenderを整数2に設定している

構造体4

例：住所録

情報（入力）：氏名，性別，住所

構造体の要素にアクセスするには

```
strcpy(pdata[0].name, "Taro");  
pdata[0].gender = 1;  
strcpy(pdata[0].address, "Ookayama, Meguro-ku, Tokyo, Japan");  
printf("Name%s, Gender(M1/F2): %d, Address. %s¥n",  
       pdata[0].name, pdata[0].gender, pdata[0].address);
```

* 後で述べるが，ポインタの場合には表記が異なる

同じ構造体であれば，代入できる

```
pdata[20]=pdata[0];
```

構造体を関数の引数や返り値に使うこともできる

（例： return pdata[0]; など）

このことはEx 3-3で利用できる

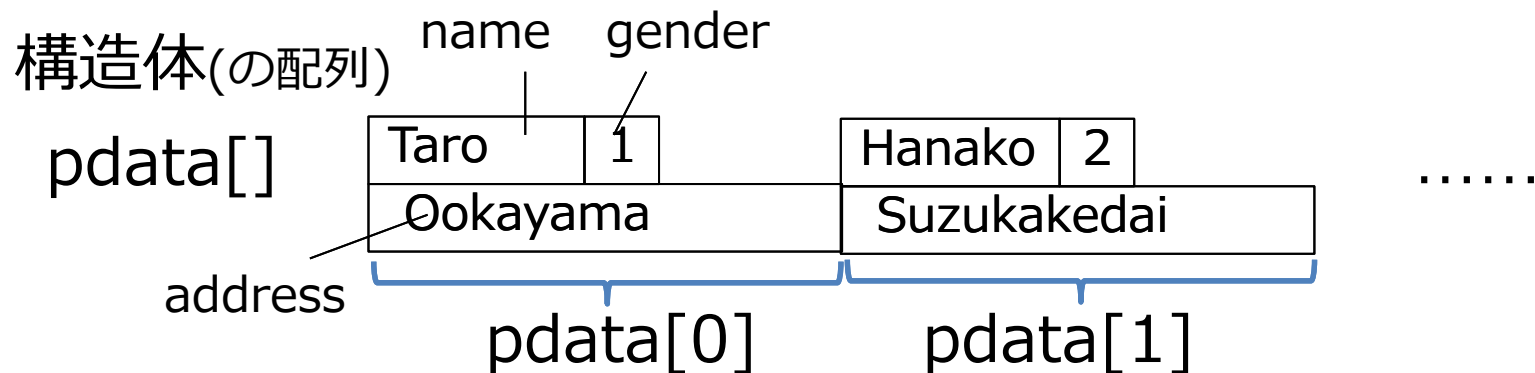
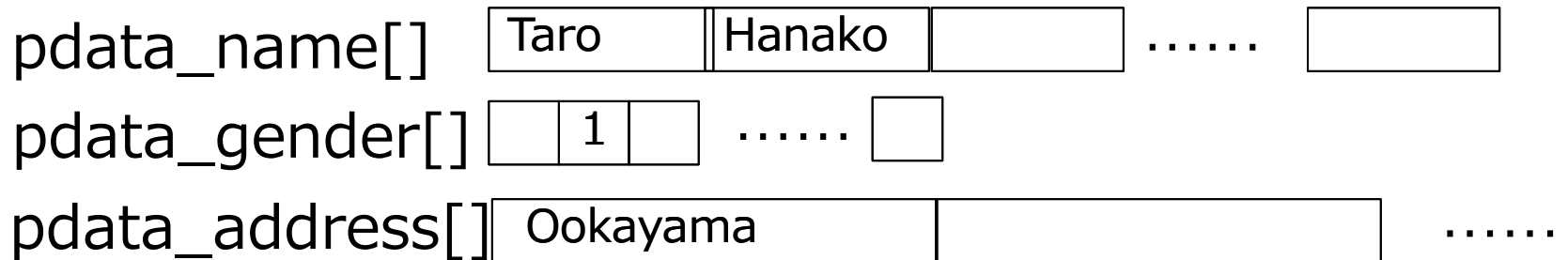
構造体5

例：住所録

情報：氏名，性別，住所

イメージ図（説明のため，実際のメモリ上のイメージとは異なる）

単純な配列：



構造体5

例：住所録

情報：氏名，性別，住所

実際のメモリ上では、

単純な配列:

pdata_name[] Taro Hanako

pdata_gender[]

	1	
--	---	--

--

pdata_address[]	Ookayama	Suzukake
-----------------	----------	----------	-------

構造体(の配列)	name	gender	address
----------	------	--------	---------

pdata[] Taro 1 Ookayama Hanako 2

pdata[0]

pdata[1]

22

構造体6(初期化)

例：住所録


情報（入力）：氏名，性別，住所

構造体の要素の初期化

```
#define N_PERSON 50
```

```
struct pdata_st {  
    char name[40];  
    int   gender;  
    char address[100];  
};
```

```
struct pdata_st pdata [N_PERSON] = { { "Taro", 1, "Ookayama, Tokyo" },  
                                       { "Hanako", 2, "Suzukakedai, Kanagawa" },  
                                       { "Jiro", 1, "Tamachi, Tokyo" }  
};
```



3つ目の要素まで初期化される

構造体6(初期化の続き)

例：住所録

情報（入力）：氏名，性別，住所

構造体の要素の初期化の別の方法

- 構造体中身の一部だけ初期化したいときに便利
//以下の例だと名前と住所だけ初期化した（性別わからないため）

```
struct pdata_st {  
    char name[40];  
    int   gender;  
    char address[100];  
};
```



```
struct pdata_st pdata [N_PERSON] = { {.name = "薫", .address="Ookayama"},  
};
```


ユーザ定義のデータ型

例：住所録

情報（入力）：氏名，性別，住所

```
#define N_PERSON 50 /* #person */
```

```
typedef struct pdata_st {  構造体の名称は省略可  
    char name[40];          /* Name */  
    int  gender;            /* Gender(M1, F2) */  
    char address[100];      /* Address */  
} pdata_t;  ユーザ定義のデータ型名称
```

```
pdata_t pdata[N_PERSON];  実際の変数（配列）
```

構造体の要素にアクセスする方法は同じ！

構造体7:ネスト(入れ子)

例：住所録

情報：氏名，性別，住所(住所，郵便番号，国名)

```
#define N_PERSON 50 /* #person */
struct address_st {
    char address[100]; /* Detailed Address */
    char zip[10];      /* Postal Code */
    char country[20];  /* Country/Region */
};
struct pdata2_st {
    char name[40]; /* Name */
    int gender;    /* Gender(1=Male, 2=Female) */
    struct address_st addr; /* Address */
};
struct pdata2_st pdata2[N_PERSON];
```

構造体7:ネスト（続き）

例：住所録

情報：氏名，性別，住所(住所，郵便番号，国名)

構造体の要素にアクセスするには

```
strcpy(pdata2[0].name, "Taro");  
pdata2[0].gender = 1;  
strcpy(pdata2[0].addr.address,  
       = "Ookayama, Meguro-ku, Tokyo";  
strcpy(pdata2[0].addr.zip, "152-0033");  
strcpy(pdata2[0].addr.country, "Japan");
```

構造体と共用体1

例：住所録

情報：氏名，性別，住所(住所，郵便番号，国名)

或いは

氏名，性別，電話番号，電子メール

```
#define N_PERSON 50 /* #person */
```

```
struct address_st { /* Surface(Physical) address */  
    char address[100]; /* Detailed Address */  
    char zip[10]; /* Postal Code */  
    char country[20]; /* Country/Region */  
};
```

```
struct e_address_st { /* Electronic address */  
    char phone [30]; /* phone number */  
    char email [50]; /* e-mail address */  
};
```

(次のページに続く)

構造体と共用体2

例：住所録

情報：氏名，性別，住所(住所，郵便番号，国名)

或いは

氏名，性別，電話番号，電子メール

(前のページからの続き)

```
struct pdata3_st {  
    char name[40];    /* Name */  
    int  gender;       /* Gender (1=Male, 2=Female) */  
    union {  
        struct address_st addr; /* Surface Address */  
        struct e_address_st eaddr;  
    };  
};  
struct pdata3_st pdata3[N_PERSON];
```

名称を省略すると, addr, eaddrが要素名になる

(説明と使い方は以降のページに)

構造体と共用体2'

例：住所録

情報：氏名，性別，住所(住所，郵便番号，国名)

或いは

氏名，性別，電話番号，電子メール

(前のページからの続き)

```
struct pdata3_st {  
    char name[40];    /* Name */  
    int  gender;      /* Gender (1=Male, 2=Female) */  
    union address_u {  
        struct address_st addr; /* Surface Address */  
        struct e_address_st eaddr;  
    } u;  
};  
struct pdata3_st pdata3[N_PERSON];
```

共用体の名称は省略しないときは

要素名が **u.addr, u.eaddr** となる

(説明と使い方は以降のページに)

構造体と共用体2''

c.f.

例：住所録

情報：住所(住所, 郵便番号, 国名)

或いは

電話番号, 電子メール

```
union address_u {  
    struct address_st addr; /* Surface Address */  
    struct e_address_st eaddr;  
};
```

```
union address_u addr_data[NPERSON];
```

のような宣言もできます。

構造体と共用体3

例：住所録

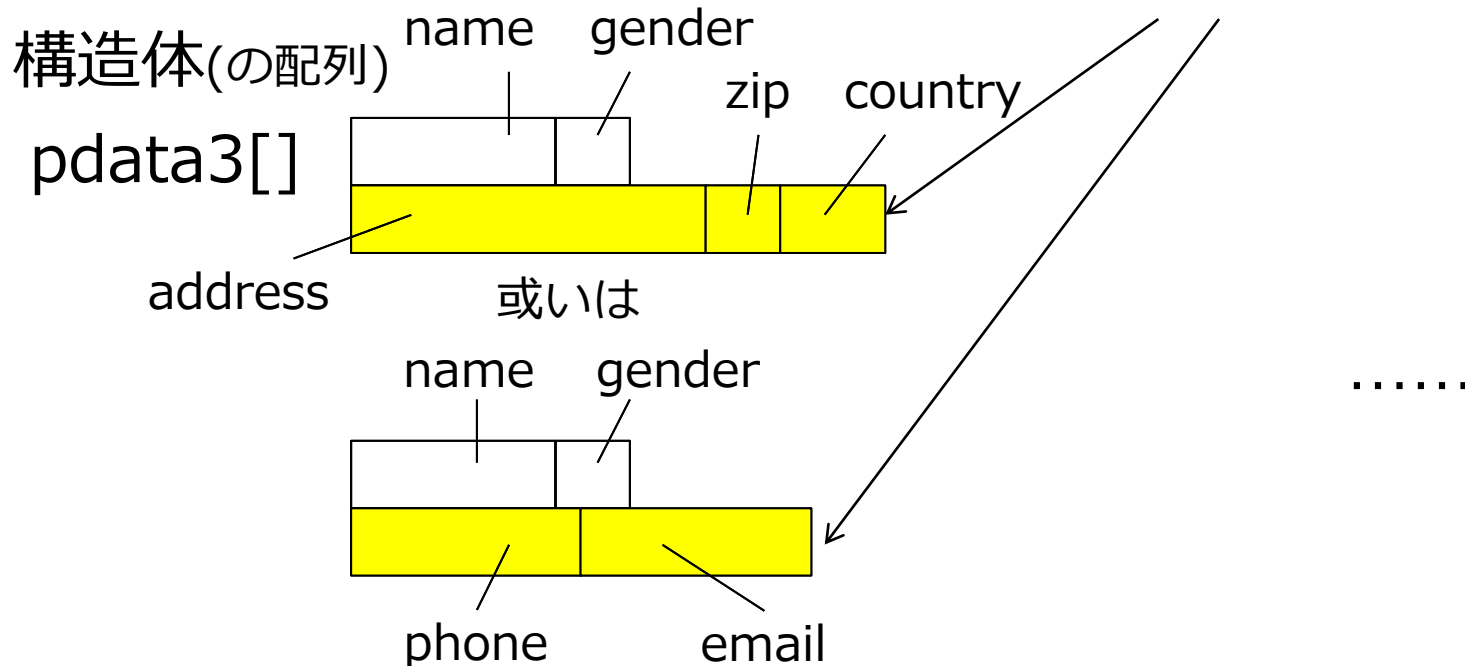
情報：氏名，性別，住所(住所，郵便番号，国名)

或いは

氏名，性別，電話番号，電子メール

(前のページからの続き)

イメージ (説明のため，実際のメモリ上のイメージとは異なる) 共通のメモリ領域
(どちらか一方しか使えません)



構造体と共用体3

例：住所録

情報：氏名，性別，住所(住所，郵便番号，国名)

或いは

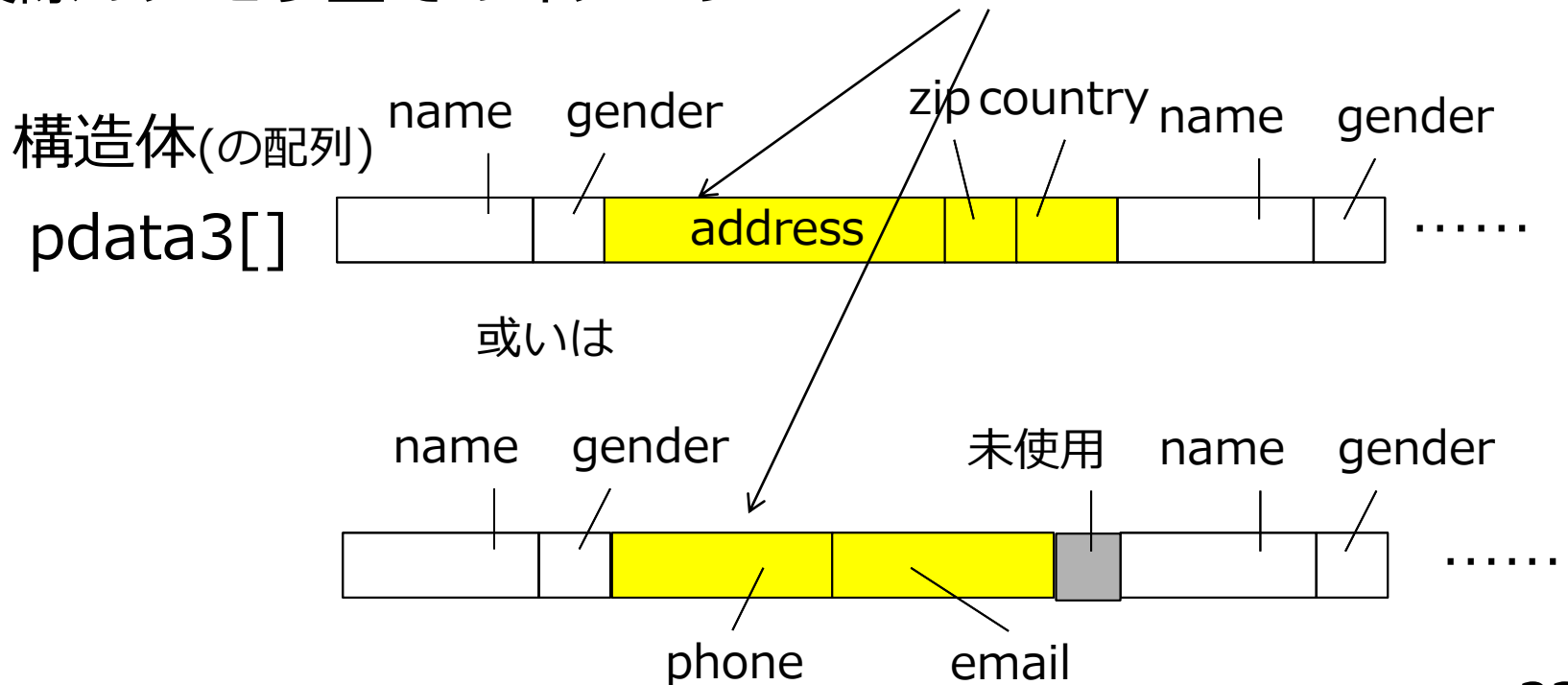
氏名，性別，電話番号，電子メール

(前のページからの続き)

共通のメモリ領域

(どちらか一方しか使えません)

実際のメモリ上でのイメージ



構造体と共用体4 (構造体と共用体 2 に対応)

例：住所録

情報：氏名，性別，住所(住所，郵便番号，国名)

或いは

氏名，性別，電話番号，電子メール

(前のページからの続き)

構造体（共用体含む）の要素にアクセスするには

```
strcpy(pdata3[0].name, "Taro");  
pdata3[0].gender = 1;  
strcpy(pdata3[0].addr.address,  
        "Ookayama, Meguro-ku, Tokyo");  
strcpy(pdata3[0].addr.zip, "152-0033");  
strcpy(pdata3[0].addr.country, "Japan");
```

```
strcpy(pdata3[1].name, "Hanako");  
pdata3[1].gender = 2;  
strcpy(pdata3[1].eaddr.phone, "03-5734-2111");  
strcpy(pdata3[1].eaddr.email, "hanako@m.titech.ac.jp"); 34
```

注意！
どちらか一方の項目
しか許されない

構造体と共用体4' (構造体と共用体 2' に対応)

例：住所録

情報：氏名，性別，住所(住所，郵便番号，国名)

或いは

氏名，性別，電話番号，電子メール

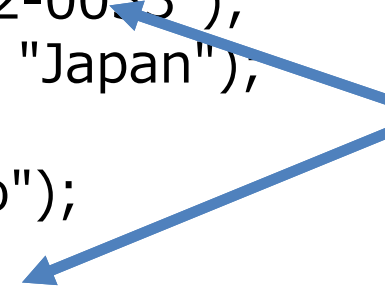
(前のページからの続き)

構造体（共用体含む）の要素にアクセスするには

```
strcpy(pdata3[0].name, "Taro");  
pdata3[0].gender = 1;  
strcpy(pdata3[0].u.addr.address,  
        "Ookayama, Meguro-ku, Tokyo");  
strcpy(pdata3[0].u.addr.zip, "152-0033");  
strcpy(pdata3[0].u.addr.country, "Japan");
```

```
strcpy(pdata3[1].name, "Hanako");  
pdata3[1].gender = 2;  
strcpy(pdata3[1].u.eaddr.phone, "03-5734-2111");  
strcpy(pdata3[1].u.eaddr.email, "hanako@m.titech.ac.jp");35
```

注意！
どちらか一方の項目
しか許されない



列挙型（ユーザ定義）

例：住所録の中の性別

```
enum _gender { male, female };
```

← int 型の0, 1, ... に
読み代えているだけ*

```
struct pdata4_st {  
    char name[40];          /* Name */  
    enum _gender gender;    /* Gender */  
    :  
};
```

```
struct pdata4_st pdata4[N_PERSON];
```

使うときは、例えば、

```
pdata4[0].gender = male;
```

```
printf("Gender %s¥n",
```

```
    (pdata4[0].gender==male ? "male" : "female"));
```

* male=1, female=2 などとすると割り当てる定数を変更できる

Ex3-1 (文字列と十進数の復習 2)

1. `int n;`と宣言した整数型変数に`scanf("%d",&n);`で非負整数を読み込み、
2. `char string[128];`と宣言した配列に十進数の文字列に変換して格納し、
3. `printf("¥n%s¥n", string);`によって出力し終了するプログラムを作成せよ

例えば、123…が入力された時には、`string[0]='1'`, `string[1]='2'`, `string[2]='3'`, …となるようにし、`printf("n%s¥n", string);`によって123…が出力されるようにする

注意事項など：

- `printf()`と`scanf()`と`fflush()`以外に開発環境が提供する関数を用いないこと
- 提出ファイル： **ex3_1.c**

Ex3-2 (構造体)

まず、構造体を使ってみよう

1) 構造体3の頁の構造体(pdata_st)により, 下記のデータを初期値として与え, 名前(Name)を入力(scanfにより取得)して線形検索したときに, 残りの情報 (Gender, Address), 或いは, 見つからない“Not Found” を出力するプログラムを作成せよ.

Name	Gender	Address
Taro	1	Ookayama, Meguro-ku, Tokyo, JAPAN
Hanako	2	Suzukakedai, Midori-ku, Yokohama, JAPAN
Jiro	1	Tamachi, Minato-ku, Tokyo, JAPAN
Ichiro	1	Miami, Florida, USA
Naomi	2	Palm Beach, Florida, USA

★提出物 : 1) のプログラム “ex3_2_1.c”

Ex3-2 (ヒント)

名前(Name)による線形検索を行うときには、
関数 `strcmp()`; (プログラミング基礎参照) を使う。
(`#include <string.h>` を忘れずに)

例 : `scanf`で入力した文字列を配列 `"inpname"` (`char inpname[NN];`)

に読み込んだあとに (NNは適当な数値) 、

```
if (strcmp(pdata[i].name, inpname) == 0 ) {  
    /* 見つかったとき (inpnameと一致) */  
}
```

(NULLではなく, 0 としないとwarningが出ます)

Ex3-2

次に、構造体のネストを使ってみよう

2) 構造体7の頁の構造体により、下記のデータを初期値として与え、名前(Name)を入力(scanfにより取得)して線形探索したときに、残りの情報(Gender, Address, Country) (ZIP Codeは無視) , 或いは、見つからない“Not Found” を出力するプログラムを作成せよ.

Name	Gender	Address	Country
Taro	1 (Male)	Ookayama, Meguro-ku, Tokyo	Japan
Hanako	2 (Female)	Suzukakedai, Midori-ku, Yokohama	Japan
Jiro	1 (Male)	Tamachi, Minato-ku, Tokyo	Japan
Ichiro	1 (Male)	Miami, Florida	USA
Naomi	2 (Female)	Palm Beach, Florida, USA	USA

★提出物： 2) のプログラム “ex3_2_2.c”
("Ichiro", "Saburo"を入力して動作確認)

Ex3-2の注意

Genderの表示は"1", "2", "male", "female"のどれかを用いること

Ex3-3 (構造体)

複素数を表すデータ型

```
typedef struct {  
    double re;    // real part (実部)  
    double im;    // imaginary part (虚部)  
} complex_number; // "complex"とするとエラー  
                  ( "math.h"での定義とかぶってしまうので)
```

を使って、

1) 複素数の加算・減算及び乗算を行う関数を作成せよ
関数名は"math.h"で定義されている関数名をさけて
cmp_add(), cmp_sub(), cmp_mul() とでもせよ.

2) 1) を利用して (j を虚数単位とする)

$$(3.0 + 4.0 j) * (3.0 - 4.0 j) - (2.0 + 1.0 j)$$

の計算結果を出力せよ (実部虚部を次ページのprintfのように出力)

★提出物: 1), 2) のプログラム `"ex3_3_2.c"` (`_1`ではない)

Ex3-3のヒント

- 複素共役を計算する例を示す

```
#include <stdio.h>
```

```
typedef struct { double re; double im; } complex_number;
```

```
/* 複素共役 */
```

```
complex_number conjugate(complex_number z)
```

```
{
```

```
    complex_number cz = { z.re, -z.im };
```

```
    return cz;
```

```
}
```

```
int main(void)
```

```
{
```

```
    complex_number z = {2.0, 1.0}, cz = conjugate(z);
```

```
    printf("%f + j * %f¥n", cz.re, cz.im);
```

```
    return 0;
```

```
}
```

Ex3-4 (文字列と16進数の復習)

1. `char string[128];`と宣言した配列に`scanf("%s", string);`を用いてキーボードから非負の**16進数**(0~9、a~f、A~Fからなる文字列)を読み込み
2. **unsigned** `int n;`と宣言した**非負**整数型変数にstringが16進数として表現する整数を代入し、
3. その値を `printf("¥n%x¥n", n);` で出力し終了するプログラムを作成せよ

注意事項など

- ステップ 2 は`atoi()`や`strtoul()`や`sscanf()`を用いると一瞬で作成できるが、問題の趣旨はそういうことではないので、この課題では`printf`と`scanf`と`fflush`以外に開発環境が提供する関数を使用しないこと
- 入力16進数の英字 (a,b,c,d,e,f) は大文字でも小文字でも同じように解釈し動作すること
- 入力の値は十進数として0以上20億以下と仮定してよい
- 入力値20億に対し正しく動作することを必ず確認して下さい
- 提出ファイル: **ex3_4.c**

%xについて

- printfやscanfで整数を16進数で表示・入力する%xは、対応する整数型が符号無しであることを要求し、符号有り整数を用いた場合の動作は規格上では未定義である
- %xに対応する変数に符号有り変数を与えても、値が非負のときには正しく動作することが多い

提出するファイル名一覧

- ex3_1.c
- ex3_2_1.c
- ex3_2_2.c
- ex3_3_2.c
- ex3_4.c
- ex3_5.c

もし時間が余ったら...

- 次回講義資料を予習し次回の課題に着手して下さい

うまく動かない質問時のお願い

- プログラムが上手く動作していないと疑わしい部分の変数を `printf` で表示するようにしてから質問して下さい（課題提出時にはデバッグ用`printf`は消す）
- （講師が松本のときのみ）質問する前に「`check1.sh`」によるコンパイラの警告と実行時エラーを無くして下さい。警告やエラーの意味がわからないときは気軽に聞いてください

コピペレポートについて

プログラムや考察などが他の提出者と重複している場合、不正とみなして減点および問い合わせをすることがあります