cp-13. 構造体

（C プログラミング入門）

URL: https://www.kkaneko.jp/pro/adp/index.html

金子邦彦

CC BY NC SA

C言語における構造体の定義方法、構造体を用いた配列・ポインタ・リストの作成と操作

【学習内容の構成】

1. **構造体の基本**：複数のデータをグループ化して新しい型を定義する方法
2. **構造体の配列**：住所録のような複数件のデータを構造体配列で管理
3. **構造体と関数**：ポインタ渡しによる構造体データの受け渡しとアロー演算子（->）によるメンバアクセス
4. **構造体のリスト**：ポインタを用いた動的なデータ構造の実現と動的メモリ管理（new/malloc）

- 前提：C言語の基本データ型、配列、ポインタ、関数の知識
- 意義：実用的なデータ管理プログラムの設計能力の獲得

内容

例題１．住所録

例題２．構造体と関数

構造体とは．構造体を扱う関数.

例題３．構造体のリスト

- 構造体とポインタの組み合わせ

目標

- 自分で「構造体」を定義する
- 自分で定義した構造体について，配列やポインタを作成する

データ型

- 基本データ型

  char　　文字（１文字）

  int　　整数

  double　　浮動小数

  など

- その他のデータ型

  配列→　データの並び（文字列も，文字の並び）

  ポインタ　→　メモリアドレス

  構造体　→　いくつかのデータを「グループ化」したもの

  など

データの集まり

住所
氏名
支店名
口座番号
残高

銀行口座

氏名
年齢
住所

住所録

実数部
虚数部

複素数

構造体とは？

- いくつかのデータを，グループ化して，新しい型の名前を付けたもの
- 基本データ型（整数，浮動小数）などの組み合わせ

例題１．住所録

- 住所録を表示するプログラムを作る
  - 住所録データを扱うために，構造体の配列を使う
  - 住所録は，名前（サイズ２０の文字の配列），年齢，住所（サイズ４０の文字の配列）から構成する

```
#include <stdio.h>
#pragma warning(disable:4996)
struct Person {
    char name[20];
    int age;
    char address[40];
};

int main()
{
    struct Person a[] = {{"Ken", 20, "NewYork"},
                         {"Bill", 32, "HongKong"},
                         {"Mike", 35, "Paris" }};
    int i;
    for ( i = 0; i < 3; i ++ ) {
        printf( "name=%s, age=%d, address=%s¥n", a[i].name, a[i].age, a[i].address );
    }
    return 0;
}
```

構造体 Person の型宣言

構造体の配列 a の宣言と初期化

構造体のメンバの読み出し

住所録

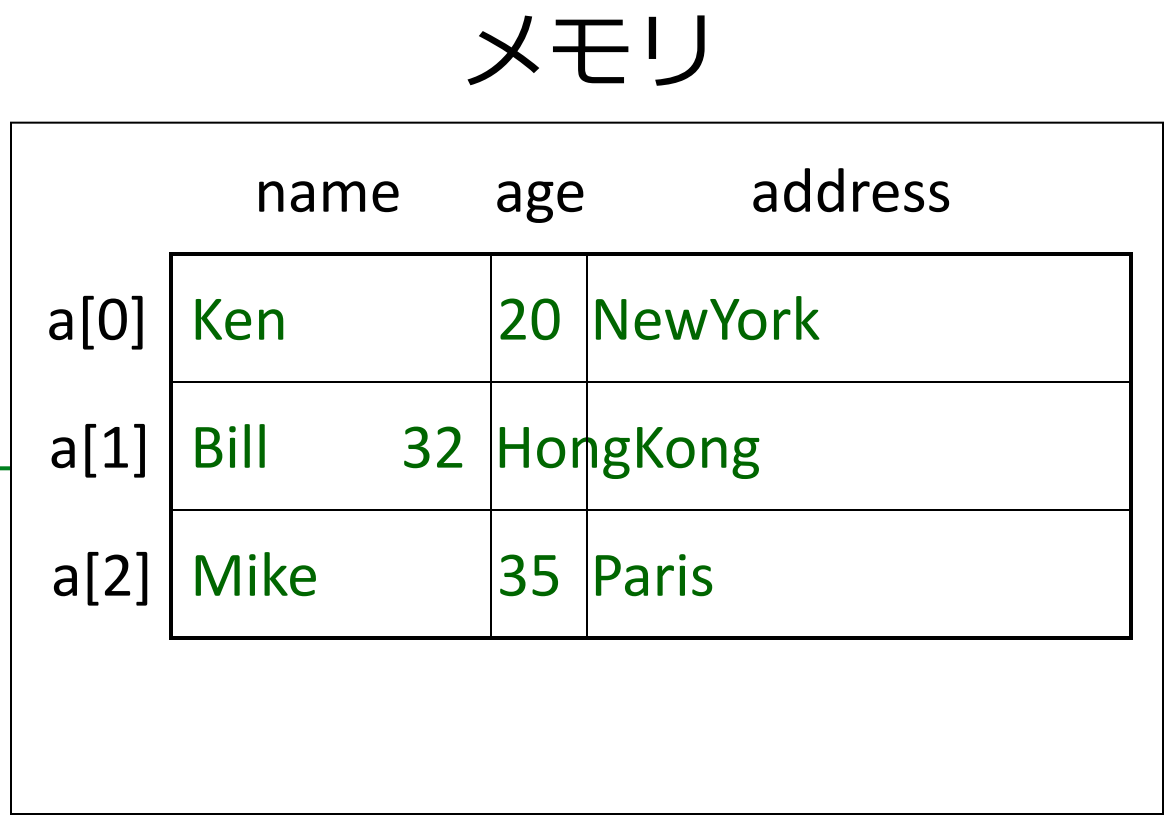
実行結果の例

```
name=Ken, age=20, address=NewYork

name=Bill, age=32, address=HongKong

name=Mike, age=35, address=Paris
```

プログラムとデータ



a[i].name

a[i].age

a[i].address

構造体の配列からの値の読み出し

構造体の型宣言

- 構造体は，いくつかのデータをグループ化したもの.
- 構造体には，名前がある
- それぞれのデータ（メンバという）は，名前と型（データの種類のこと）がある.

```
struct Person {
    char name[20];
    int age;
    char address[40];
};
```

名前

メンバ

構造体メンバの読み書き

	name	age	address
a[0]	Ken	20	NewYork
a[1]	Bill 32	HongKong	
a[2]	Mike	35	Paris

- 配列の中身を読み書きするときには，構造体のメンバを書く

例） a[i].name

これがメンバ

構造体の使い方

```
#include <stdio.h>
#pragma warning(disable:4996)
struct Person {
    char name[20];
    int age;
    char address[40];
};

int main()
{
    struct Person a[] = {{"Ken", 20, "NewYork"},
                         {"Bill", 32, "HongKong"},
                         {"Mike", 35, "Paris" }};
    int i;
    for ( i = 0; i < 3; i ++ ) {
        printf( "name=%s, age=%d, address=%s¥n", a[i].name, a[i].age, a[i].address );
    }
    return 0;
}
```

①ここで「構造体　Person 」を宣言
（構造体の宣言）

②ここで　Person を使って、「構造体の配列 a」　を宣
（変数の宣言）

③ここで a を使う

例題２．構造体と関数

- ３人分の住所録を読み込んで，構造体の配列に格納した後に，表示するプログラムを作る
  - 住所録データを扱うために，構造体の配列を使う
  - 住所録は，例題１と同じく，名前（サイズ２０の文字の配列），年齢，住所（サイズ４０の文字の配列）から構成する
  - ここでは，練習のため，１人分の住所録を読み込む関数，１人分の住所録を表示する関数を作る．これら関数への引数として，構造体のポインタを渡すこと．

```
#include <stdio.h>
struct Person {
    char name[20];
    int age;
    char address[40];
};
void read_person( struct Person* a )
{
    printf("name=");
    scanf("%s", a->name );
    printf("age=");
    scanf("%d", &a->age );
    printf("address=");
    scanf("%s", a->address );
    return;
}
void print_person( struct Person* a )
{
    printf( "name=%s, age=%d, address=%s¥n", a->name, a->age, a->address );
    return;
}
```

構造体 Person の型宣言

構造体の
ポインタ渡しに関係する

```
int main()
{
    struct Person a[3];
    int i;
    for ( i = 0; i < 3; i ++ ) {
        read_person( &a[i] );
    }
    for ( i = 0; i < 3; i ++ ) {
        print_person( &a[i] );
    }
    return 0;
}
```

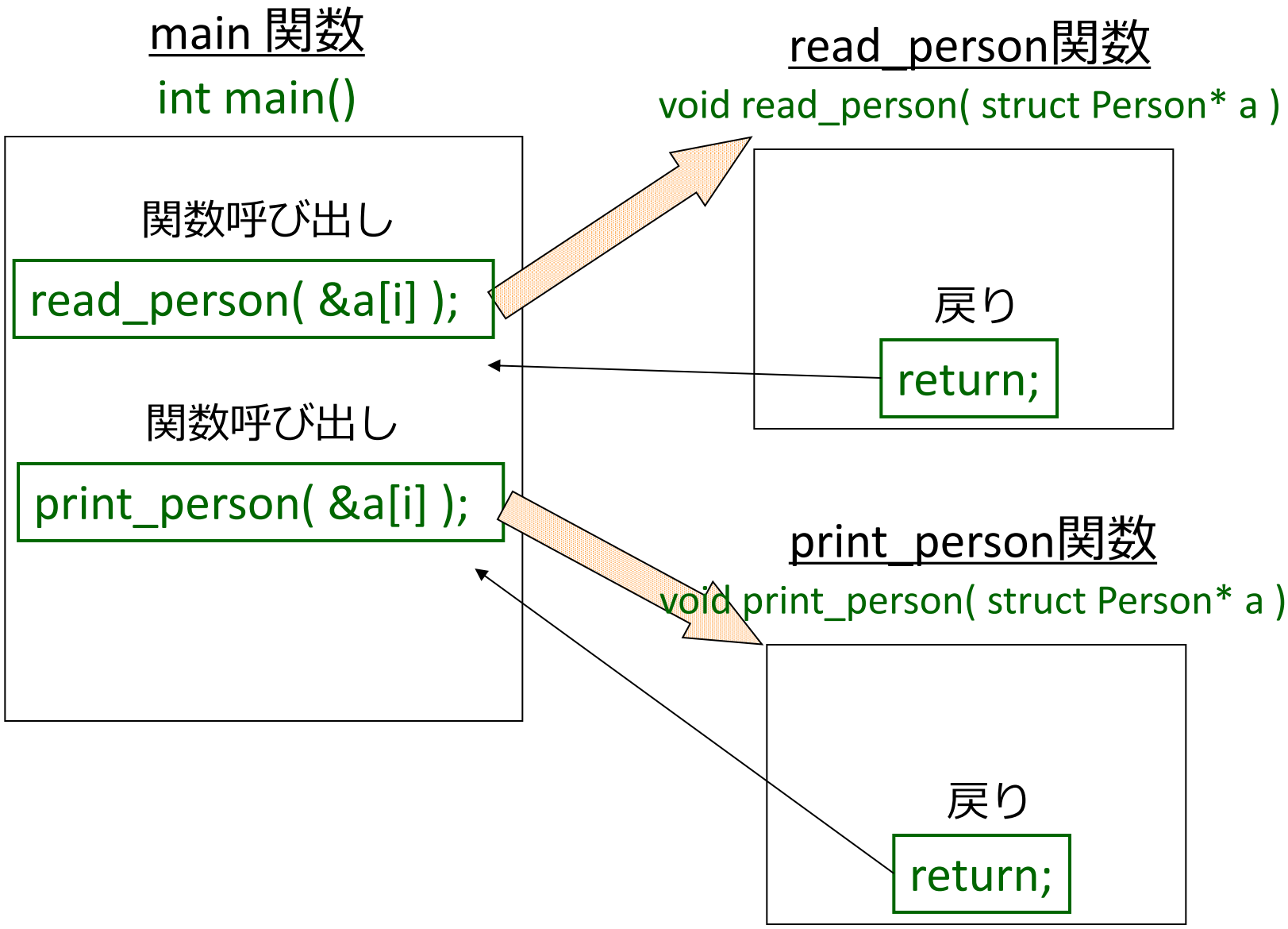
構造体の
ポインタ渡しに関係する

構造体と関数

実行結果の例

```
name=Ken
age=20
address=NewYork
name=Bill
age=32
address=HongKong
name=Mike
age=35
address=Paris
name=Ken, age=20, address=NewYork
name=Bill, age=32, address=HongKong
name=Mike, age=35, address=Paris
```

関数呼び出しの流れ


main 関数
int main()
関数呼び出し
read_person(&a[i]);
関数呼び出し
print_person(&a[i]);
read_person関数
void read_person(struct Person* a)
戻り
return;
print_person関数
void print_person(struct Person* a)
戻り
return;

プログラム実行順

```
void read_person( struct Person* a )
{
③ printf("name=");
④ scanf("%s", a->name );
⑤ printf("age=");
⑥ scanf("%d", &a->age );
⑦ printf("address=");
⑧ scanf("%s", a->address );
⑨ return;   戻り
}
```

read_person関数

```
void print_person( struct Person* a )
{
⑫ printf( "name=%s, age=%d, address=%s¥n", a->name, a->age, a->address );
⑬ return;   戻り
}
```

print_person関数

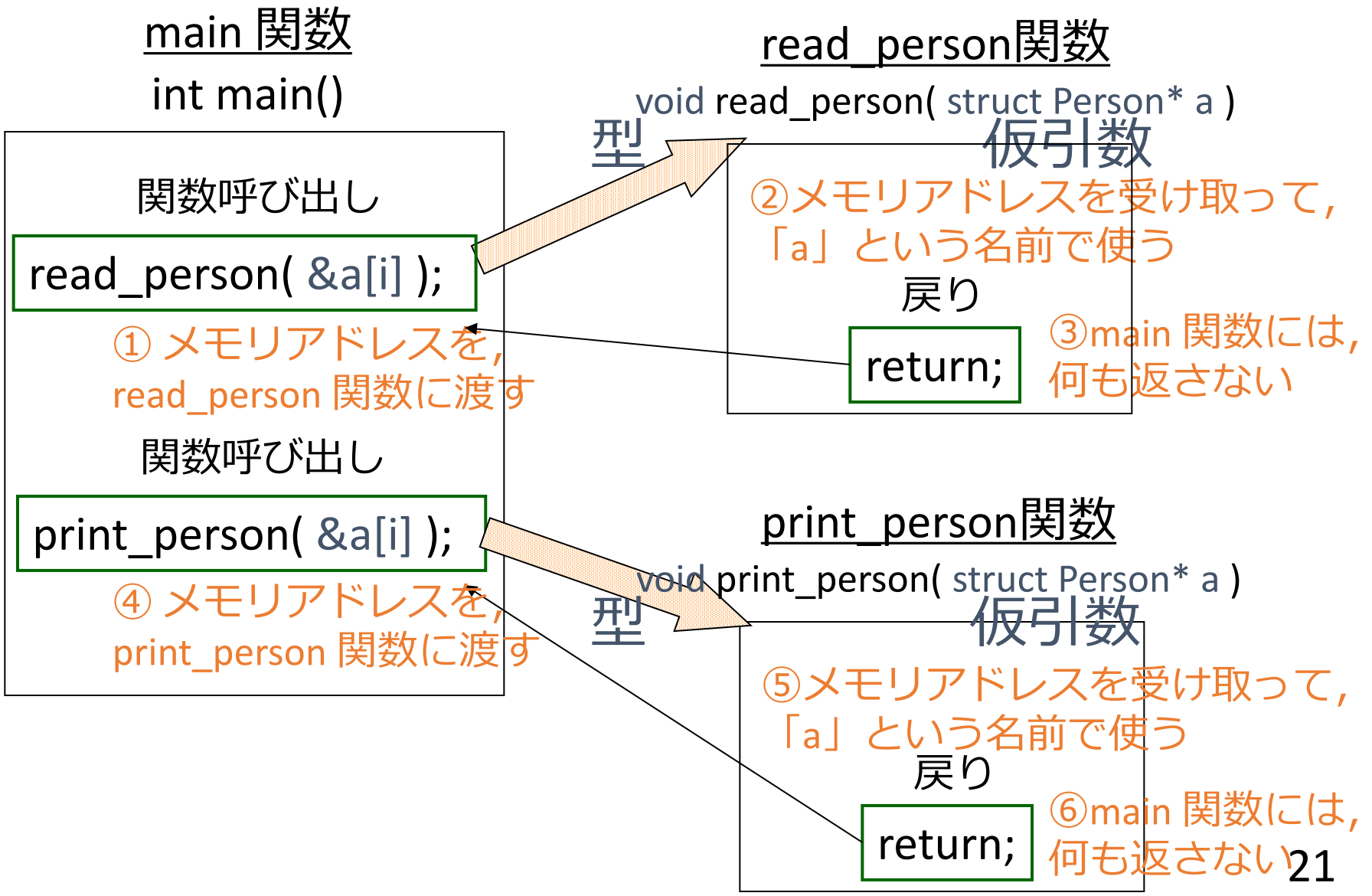
```
int main()
{
    struct Person a[3];
    int i;
① for ( i = 0; i < 3; i ++ ) {
②     read_person( &a[i] );   関数呼び出し
  }
⑩ for ( i = 0; i < 3; i ++ ) {
⑪     print_person( &a[i] );   関数呼び出し
  }
⑭ return 0;
}
```

main関数

main 関数の先頭行
がプログラムの始まり

main 関数内の return
がプログラムの終わり

データの流れ


main 関数
int main()
関数呼び出し
read_person(&a[i]);
① メモリアドレスを，
read_person 関数に渡す
関数呼び出し
print_person(&a[i]);
④ メモリアドレスを，
print_person 関数に渡す
read_person関数
void read_person(struct Person* a)
型
仮引数
②メモリアドレスを受け取って，
「a」という名前で使う
戻り
return;
③main 関数には，
何も返さない
print_person関数
void print_person(struct Person* a)
型
仮引数
⑤メモリアドレスを受け取って，
「a」という名前で使う
戻り
return;
⑥main 関数には，
何も返さない

read_person関数呼び出しでのデータの流れ

	name	age	address
a[0]	Ken	20	NewYork
a[1]	Bill	32	HongKong
a[2]	Mike	35	Paris

main 関数内で宣言された a

a メモリアドレス

	name	age	address
a[0]	Ken	20	NewYork
a[1]	Bill	32	HongKong
a[2]	Mike	35	Paris

main 関数内で宣言された a と，仮引数で宣言された a は別のもの

main 関数

int main()

関数呼び出し

```
read_person( &a[i] );
```

① メモリアドレスを，read_person 関数に渡す

read_person関数

void read_person(struct Person* a)

型　仮引数

②メモリアドレスを受け取って，「a」という名前で使う

戻り

```
return;
```

③main 関数には，何も返さない

関数への構造体の受け渡し

- 呼び出し側
  - 「＆」を付けて，メモリアドレスを，関数に渡す

例） read_person(&a[i]);

print_person(&a[i]);

「a[i] のメモリアドレス」という意

- 関数側
  - メモリアドレスを受け取ることを宣言しておく

例） void read_person(struct Person* a)

void print_person(struct Person* a)

「メモリアドレスを受け取って，a として使う」という意味

配列とポインタ

プログラム例： read_person(&a[i]);

	name	age	address	
a[0]	Ken	20	NewYork	i = 0 ならここ
a[1]	Bill	32	HongKong	i = 1 ならここ
a[2]	Mike	35	Paris	i = 2 ならここ

演算子 -> の意味

メモリアドレスから，構造体メンバにアクセス

例）

```
void read_person( struct Person* a )
{
    printf("name=");
    scanf("%s", a->name );
    printf("age=");
    scanf("%d", &a->age );
    printf("address=");
    scanf("%s", a->address );
    return;
}
```

ここでは，「a」に入っているのはメモリアドレス

scanf で a->age にだけ & をつける理由

```
scanf("%s", a->name );
scanf("%d", &a->age );
scanf("%s", a->address );
```

- 文字列
  - a->name, a->address は，「文字の配列の先頭メモリアドレス」という意味　（＆は付けない）
- 整数，浮動小数
  - &a->age は，「整数データのメモリアドレス」という意味（＆を忘れると，うまく動かない）

課題１．住所録の条件検索

- ３人分の住所録を読み込んで，構造体の配列に格納した後に，「２０歳以上」のデータだけを選んで表示するプログラムを作りなさい
  - ここでは，「２０歳以上のデータだけを表示する機能」を持った関数(main 関数とは別の関数)を作ること
  - 住所録データを扱うために，構造体の配列を使うこと
  - 住所録は，例題１と同じく，名前（サイズ２０の文字の配列），年齢，住所（サイズ４０の文字の配列）から構成する
  - 確かに，２０歳以上のデータだけが表示されることを確認すること

課題２．日付

- 日付を扱う構造体を設計し、それを使ったプログラムを作成しなさい
  - 日付データを扱うために，構造体を使うこと
  - 日付は，年（整数データ），月（整数データ），日（整数データ）から構成する
  - 日付を読み込んで，1か月分のカレンダーを表示するようなプログラムであること.

    例）日付が「２００１年１２月２１日」なら，２００１年１２月の１か月分のカレンダーを表示する

例題３．構造体のリスト

- 住所録の読み込みと表示を行うプログラムを作る
  - 住所録データを扱うために，構造体のリストを使う
  - 住所録は，名前（サイズ２０の文字の配列），年齢，住所（サイズ４０の文字の配列）から構成する
  - メニュー機能を作るために switch 文を使う

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

struct Person {
    char name[20];
    int age;
    char address[40];
};

struct PersonNode {
    struct Person person;
    struct PersonNode* next;
};

struct PersonList {
    struct PersonNode* top;
};
```

構造体 Person の型宣言

構造体 PersonNode の型宣言

構造体 PersonList の型宣言

```
void insert_head( struct PersonList* a, char* name, int age, char* address )
{
    struct PersonNode* x = new struct PersonNode();
    strcpy(x->person.name, name);
    x->person.age = age;
    strcpy(x->person.address, address);
    x->next = a->top;
    a->top = x;
    return;
}
void input_data( struct PersonList* a )
{
    char name[20];
    int age;
    char address[40];
    printf("name=");
    scanf("%s", name );
    printf("age=");
    scanf("%d", &age );
    printf("address=");
    scanf("%s", address );
    insert_head( a, name, age, address );
    return;
}
```

構造体へのポインタ x の宣言と初期化

構造体のメンバの読み出し

構造体のメンバの読み出し

```
void print_person( struct Person* a )
{
    printf( "name=%s, age=%d, address=%s¥n", a->name, a->age, a->address );
    return;
}
void print_data( struct PersonList* a )
{
    PersonNode* current;
    if ( a->top == NULL ) {
        return;
    }
    current = a->top;
    do {
        print_person( &(current->person) );
        current = current->next;
    } while( current != NULL );
}
```

構造体へのポインタ current の宣言

構造体のメンバの読み出し

```
void menu()
{
    struct PersonList* a = new PersonList();
    a->top = NULL;
    int command;
    while(1) {
        printf("menu¥n");
      printf("--------¥n");
      printf("1. input¥n");
      printf("2. print¥n");
      printf("9. exit¥n");
      scanf("%d", &command );
        if ( command == 9 ) {
        return;
      }
      switch( command ) {
        case 1:
            input_data( a );
            break;
        case 2:
            print_data( a );
            break;
        default:
            printf("invalid command¥n");
      }
    }
    return;
}
```

構造体へのポインタ a の宣言と初期化

switch 文については後述

```
int main()
{
    menu();
    return 0;
}
```

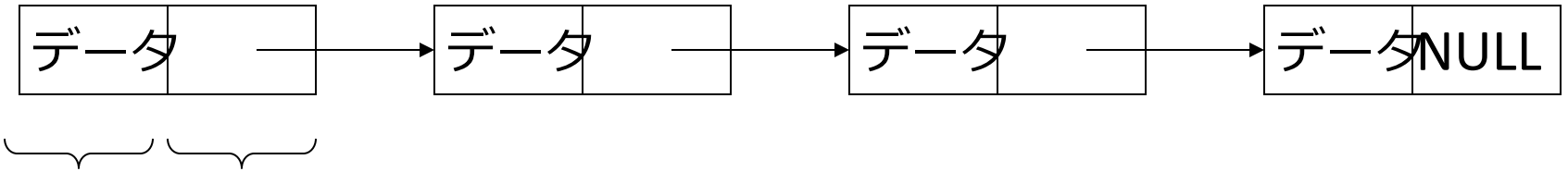
実行結果の例

```
menu
--------
1. input
2. print
9. exit
1
name=Ken
age=20
address=NewYork
menu
--------
1. input
2. print
9. exit
1
name=Bill
age=32
address=HongKong
menu
--------
1. input
2. print
9. exit
2
name=Bill, age=32, address=HongKong
name=Ken, age=20, address=NewYork
```

リスト

- 「何か」を順に並べたもの
- 順序に意味がある

ポインタを使ったリストの実現例



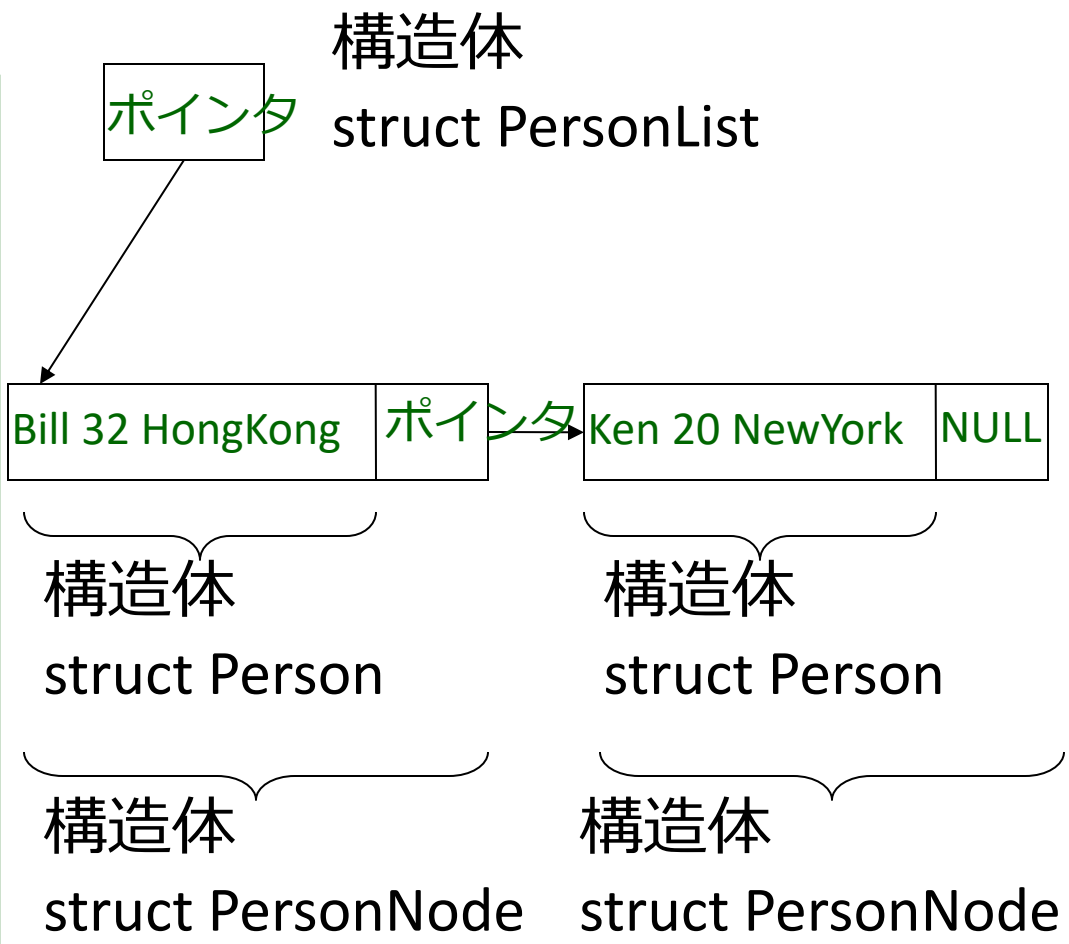
データ部分　ポインタ部分

1つの構造体

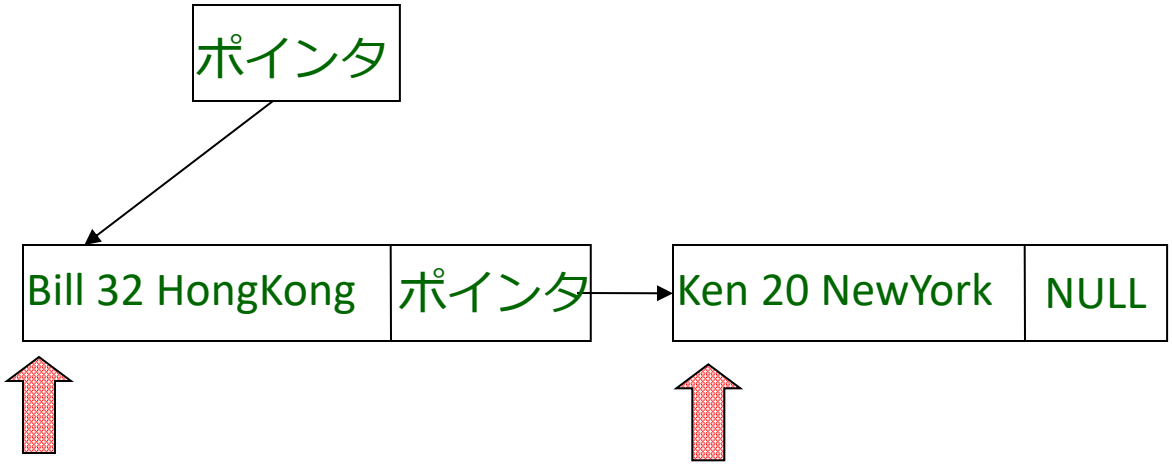
NULLで，リストの末端であることを表す

リストの例

```
struct Person {
    char name[20];
    int age;
    char address[40];
};
struct PersonNode {
   struct Person person;
   struct PersonNode* next;
};
struct PersonList {
   struct PersonNode* top;
};
```



リストの辿り



ループ 1 回目は，cuurent は，ここへのポインタ

ループ 2 回目は，cuurent は，ここへのポインタ
(current->next が NULL なので，ループが終了する)

```
current = a->top;
do {
        print_person( &(current->person) );
        current = current->next;
} while( current != NULL );
```

動的メモリ管理

- プログラムの実行中に、必要に応じて「メモリを確保」、「メモリを解放」すること

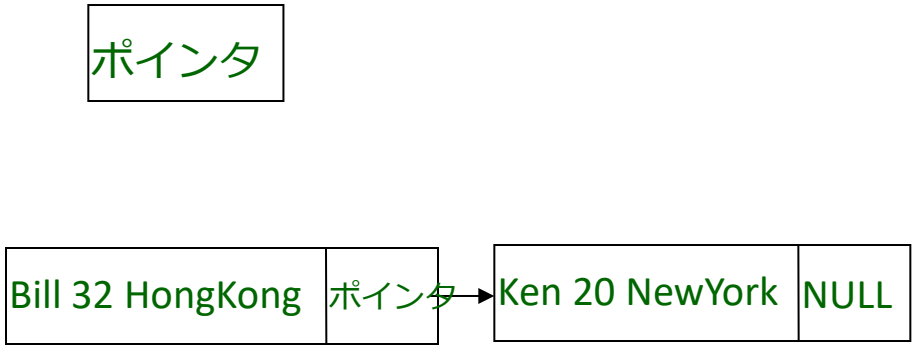
- new, delete を使用

リストと動的メモリ管理

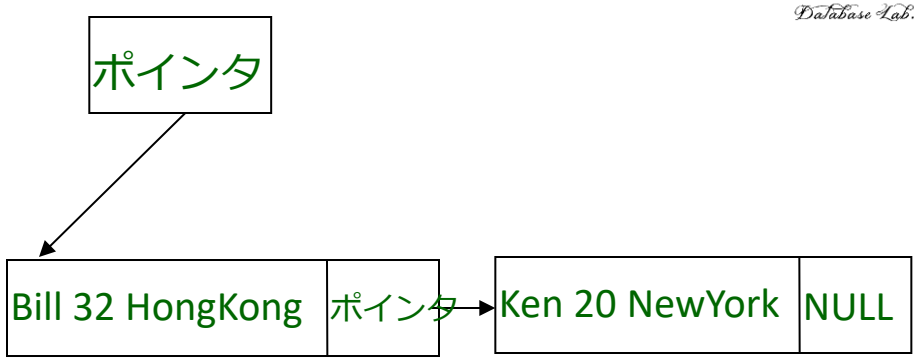
「新しいノード」を作るには，new あるいは malloc を使う．下記は new を使った例

```
void insert_head( struct PersonList* a, char* name, int age, char* address )
{
    struct PersonNode* x = new struct PersonNode();
    strcpy(x->person.name, name);
    x->person.age = age;
    strcpy(x->person.address, address);
    x->next = a->top;
    a->top = x;
    return;
}
```

挿入



⑤の時点



⑥の時点

```
void insert_head( struct PersonList* a, char* name, int age, char* address )
{
① struct PersonNode* x = new struct PersonNode();
② strcpy(x->person.name, name);
③ x->person.age = age;
④ strcpy(x->person.address, address);
⑤ x->next = a->top;
⑥ a->top = x;
   return;
}
```

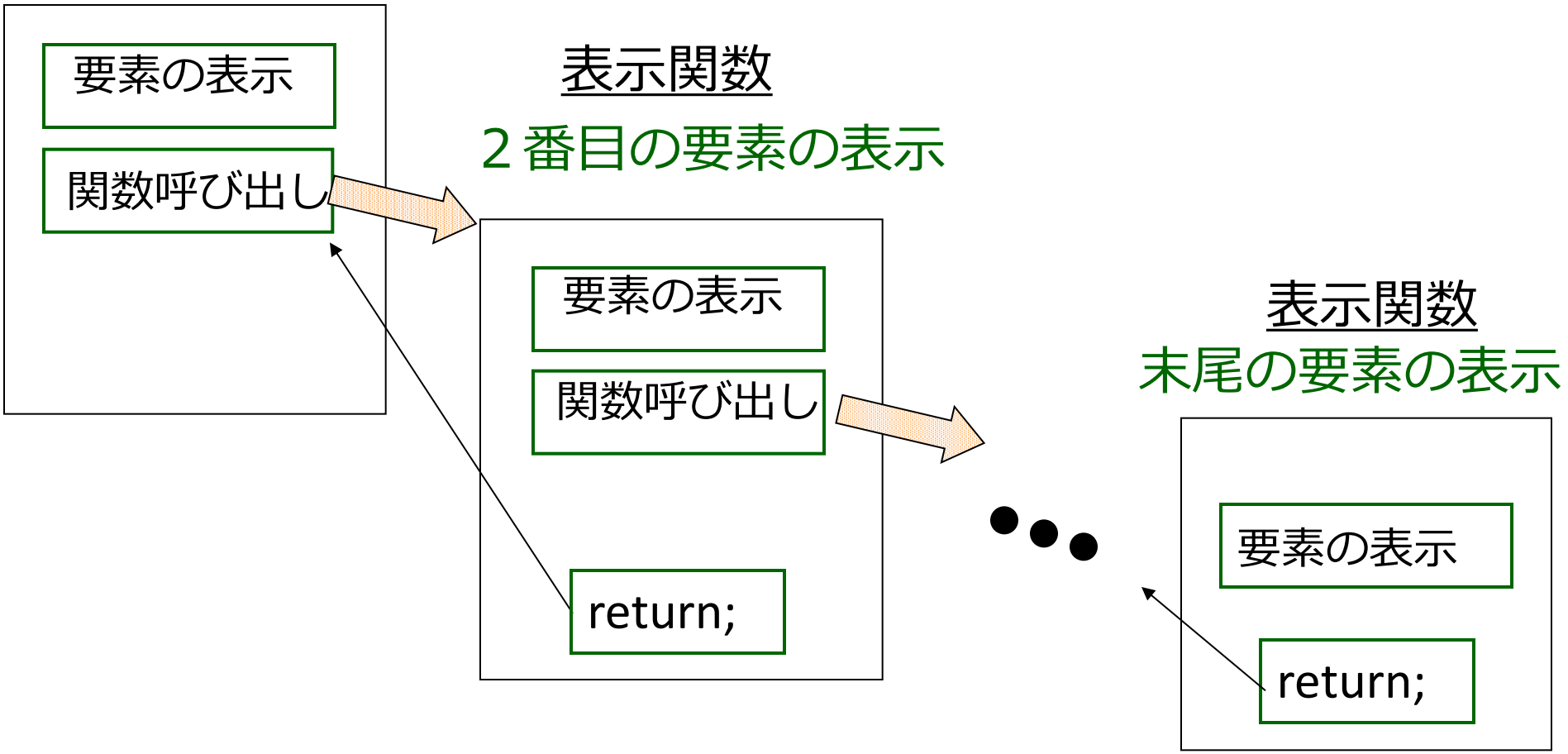
動的メモリ管理のメリット

- 必要な分だけのメモリを、好きなときに得られる
  1. 「リスト」は，必要に応じて，大きくなったり小さくなったりする
  2. 配列は，あらかじめサイズが決まっていて，サイズを超えるデータは入らない

課題３．住所録

- 例題３のプログラムについて，住所録の表示を関数の再帰呼び出しによって行うように書き換えなさい.

課題３のヒント

表示関数

１番目の要素の表示