

プログラミング基礎

<http://bit.ly/prog2d>

構造体の配列とCSVファイル

後期 第11週

2017/12/11

「struct」で構造体型の**宣言**
「typedef」で型の**別名を定義**
(構造体の宣言は、関数のプロトタイプ
宣言よりも前ですることに注意)

```
1: #include <stdio.h>
2: #define NUM 3
3:
4: typedef struct Car {
5:     int num;
6:     double gas;
7: } Car;
8:
9: void show_cars(Car *pC, char *str);
10: void save_cars(Car *pC);
11: void load_cars(Car *pC);
12:
```

【Point 1】 構造体型配列の宣言は、int型やchar型の配列と同様に変数名の後に要素数を指定する

```
13: int main(void)
14: {
15:     Car cars1[3], cars2[3];
16:     int i;
17:
18:     cars1[0].num = 1234; cars1[0].gas = 25.5;
19:     cars1[1].num = 4567; cars1[1].gas = 52.2;
20:     cars1[2].num = 7890; cars1[2].gas = 20.5;
21:
22:     show_cars(cars1, "cars1");
23:
24:     return 0;
25: }
```

【Point 2】 配列の各要素のメンバに値を代入する

【Point 3】 構造体配列の先頭アドレス（つまり「&cars1[0]」と同じ意味）を参照渡ししている

```
27: void show_cars(Car *pC, char *str)
28: {
29:     int i = 0;
30:     printf("--- %s ---\n", str);
31:     for(i=0; i<NUM; i++) {
32:         printf("[%d] num: %d, gas: %lf\n",
33:             i, (pC+i)->num, (pC+i)->gas);
34:
35:     }
36: }
```

【Point 4】 ポインタpCが参照
している配列のi番目の要素

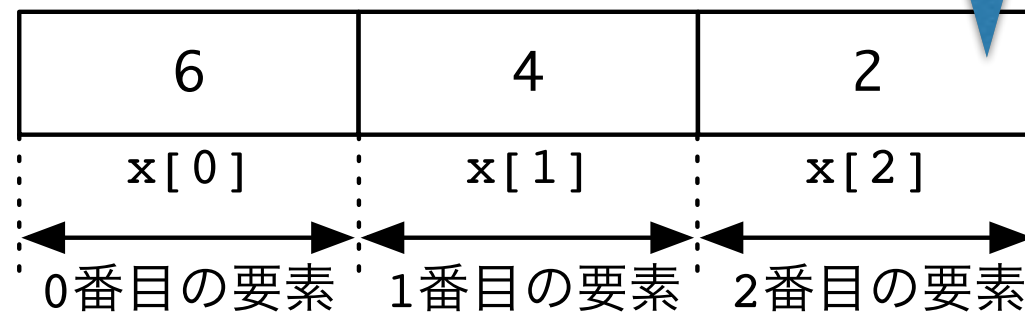
【Point 4】 アロー演算子の前のポインタが参照している要素
(つまり「pCが参照しているi番目」) のメンバnum

【Point 1】の補足

構造体型の配列は、**各要素が構造体**となっています

int型の配列の場合

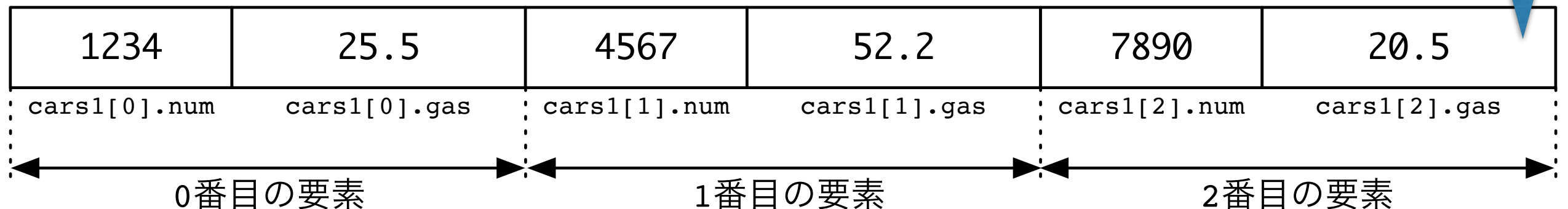
`int x[3] = {6, 4, 2};` **int型**の要素が3個用意される



構造体型の配列の場合

`Car cars1[3];`

Car型 (struct Car型) の要素が3個用意される



※ 20行目までの代入処理後の様子

【Point 1】の補足

構造体型の配列も、宣言時に初期値を与えることができます

int型の配列の場合

```
int x[3] = {6, 4, 2};
```

配列の0番目に6が代入される

構造体型の配列の場合

```
Car cars1[3] = {  
    {1234, 25.5},  
    {4567, 52.2},  
    {7890, 20.5}  
};
```

配列の0番目のメンバnumに1234、メンバgasに25.5が代入される

【Point 2】 の補足

構造体型配列の各要素のメンバを参照する場合は、**添字演算子**（`[]`）と**ドット演算子**（`.`）を組み合わせ
て使います。

`cars1[0].num = 1234`

まず、添字演算子で配列cars1
の0番目を指定する

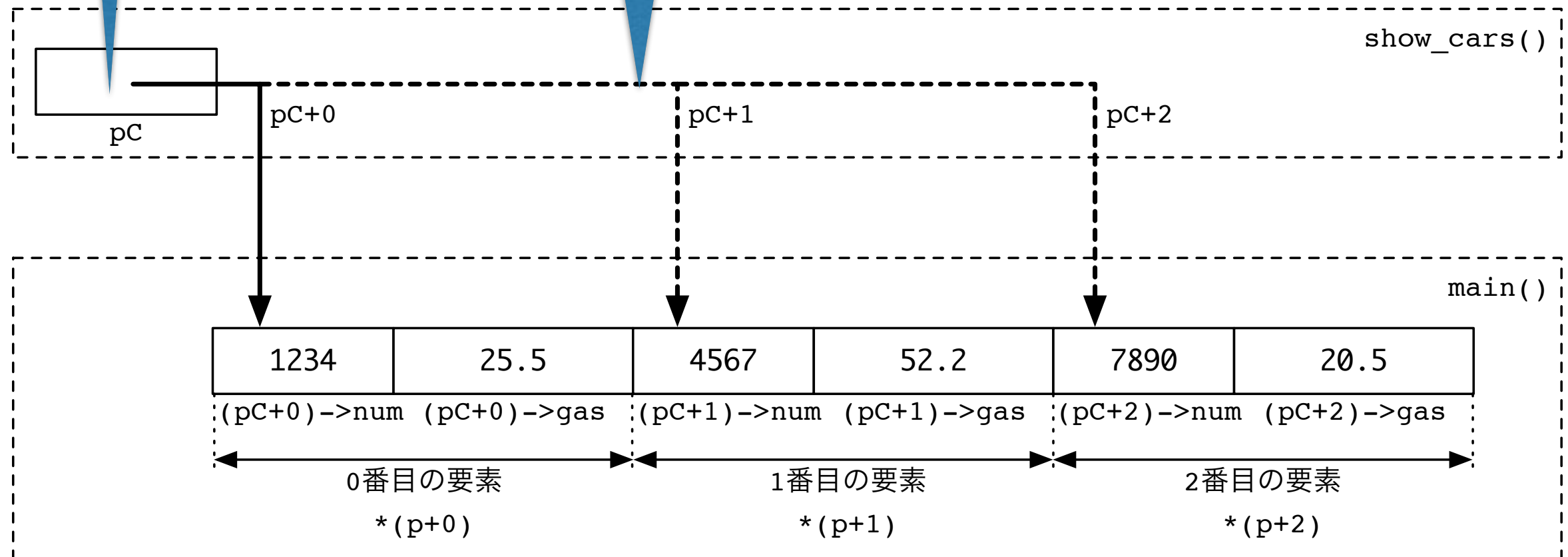
次に、ドット演算子で（0番目の）
メンバnumを指定する

【Point 3】の補足

構造体型の配列を引数で渡す場合も、int型の配列と同様に、**先頭要素のアドレスが参照渡しされます。**

main内にある配列cars1の先頭要素（0番目の要素）のアドレスが、ポインタpCへ参照渡しされる

ポインタを**1加算**すると「**1つ後の要素**」を参照する



【Point 4】の補足

ポインタが参照している配列に対して、「i番目の要素のメンバ」を表すには、**アドレスの加算**と**アロー演算子**を組み合わせで表現します。

$(pC+i) \rightarrow num$

まず、配列の先頭アドレスにiを加算する
(pCが参照している配列の先頭からi番目の要素)

次に、アロー演算子で、ポインタが参照している
要素のメンバにアクセスする

【Point 4】の補足

アロー演算子を使わない表現もできます。

間接参照演算子とドット演算子を使う場合

間接参照演算子で
「i番目」を表す

$(* (pC + i))$ $\cdot num$

ドット演算子で「メンバnum」を表す

添字演算子とドット演算子を使う場合

$pC [i]$ $\cdot num$

ドット演算子で「メンバnum」を表す

ポインタの参照 (*) とアドレスの加算 (+) の
組合せは、添字演算子に置き換えられる (後期第4週参照)

CSVファイルとして入出力する

CSV (Comma-Separated Values) は、いくつかのデータをカンマ (,) で区切ったテキスト形式で、Excelなど広く一般的に使われています。

ファイルの中身

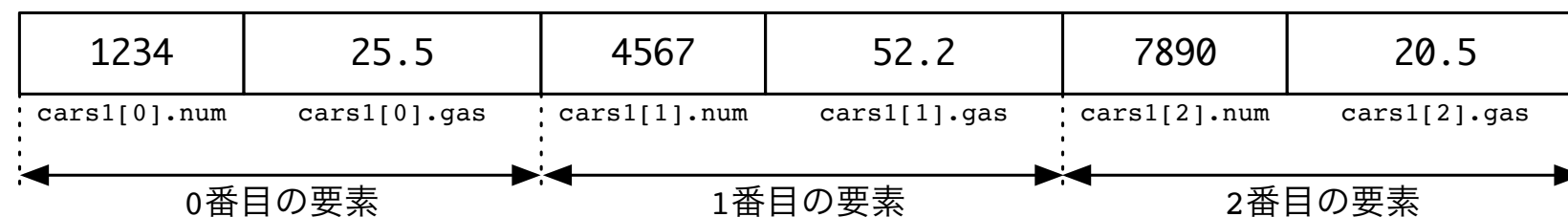
```
1234,25.500000  
4567,52.200000  
7890,20.500000
```

データが1つずつ
カンマで区切られて
いる

保存



読込



構造体の情報をファイルに対して保存/読込する際に、CSVファイルを利用できます。

CSV形式で保存する関数

【例1】 サンプルプログラムに以下の関数を追加する

```
void save_cars(Car *pC)
{
    FILE *fp;
    int i;
    fp = fopen("cars.csv", "w"); /* ファイルを書き込みモードで開く */
    if(fp == NULL) {
        printf("ファイルが開けませんでした。 \n");
        return;
    }
    printf("(saving)\n");
    for(i=0; i<NUM; i++) {
        /* pCが参照している配列のi番目の要素を、
           カンマで区切った書式でファイルへ書き込む */
        fprintf(fp, "%d,%lf\n", (pC+i)->num, (pC+i)->gas);
    }
    fclose(fp);
}
```

「カンマで区切られた1行」という書式を指定

CSV形式で保存する

【例2】 サンプルプログラムのmain()の最後に、
以下の関数呼び出しを追加する

```
show_cars(cars1, "cars1");  
save_cars(cars1);  
  
return 0;
```

配列cars1の全要素をCSV形式でファイルに保存するために、
関数save_cars()にcars1を参照渡しする

CSV形式で読み込む関数

【例3】 サンプルプログラムに以下の関数を追加する

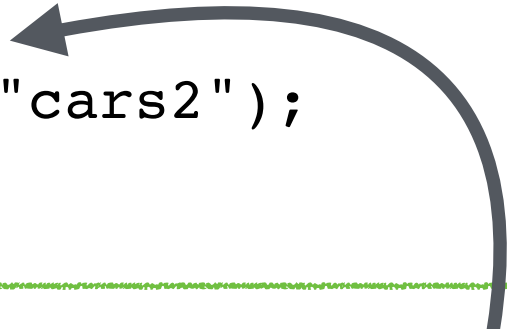
```
void load_cars(Car *pC)
{
    FILE *fp;
    int i;
    fp = fopen("cars.csv", "r"); /* ファイルを読み込みモードで開く */
    if(fp == NULL) {
        printf("ファイルが開けませんでした。 \n");
        return;
    }
    printf("(loading)\n");
    for(i=0; i<NUM; i++) {
        /* カンマで区切った書式でファイルから1行読み込み、
           構造体へ格納する */
        fscanf(fp, "%d,%lf\n", &(pC+i)->num, &(pC+i)->gas);
    }
    fclose(fp);
}
```

「カンマで区切られた1行」という書式を指定

CSV形式で読み込む

【例4】 `save_cars()`で追加した処理の代わりに、
以下の関数呼び出しを追加する

```
load_cars(cars2);  
show_cars(cars2, "cars2");  
  
return 0;
```



CSVファイルから読み込んだデータを配列cars2に格納するために、関数load_cars()にcars2を参照渡しする

【練習11-1】

サンプルプログラムを入力して、実行結果を確認してみましよう。

【準備】

関数save_cars()とload_cars()が予め入力されたファイルを、以下のようにcpコマンドを使って、自分の所にコピーしましょう。

```
$ cp /usr/local/common/kogai/prog2d/kiso211-1.c . (←ここにピリオド)
```

【練習11-2】

例1, 例2で示したように、関数save_cars()の定義を追加し、main()に関数呼び出し処理を追加して、プログラムの実行結果を確認してみましょう。

実行後に、書き出したファイル「cars.csv」を、geditなどのテキストエディタで開き、**以下のように保存されていることを確認しましょう。**

ファイルcars.csvの中身

```
1234,25.500000  
4567,52.200000  
7890,20.500000
```

【練習11-3】

例3, 例4で示したように、関数load_cars()の定義を追加し、main()に関数呼び出し処理を追加して、プログラムの実行結果を確認してみましょう。

練習11-2で保存された（つまりcar1の配列が保存された）

「cars.csv」を読み込んだ場合、cars2はcars1と同じ情報が格納されています。

geditなどのテキストエディタで「cars.csv」を開き、**手動で値の一部を変更した後に**プログラムを実行し、cars2にその値が格納されているのか確認しましょう。

【課題11-1】

構造体型struct Carの配列に対して、「指定した要素のメンバgasを加算する」関数add_gas()を作成して下さい。

[この関数のプロトタイプ宣言]

```
void add_gas(Car *p, int i, float g);
```

```
/* 引数のポインタpが参照している配列のi番目の要素に対して、  
   メンバgasに引数gを加算する */
```


【課題11-1】

[mainでの処理]

```
Car cars3[3] = {  
    {1234, 25.5},  
    {4567, 52.2},  
    {7890, 20.5}  
};  
add_gas(cars3, 1, 5.5);  
add_gas(cars3, 2, 8.2);  
show_cars(cars3, "add_gas: cars3");
```

[実行結果]

```
--- add_gas: cars3 ---  
[0] num: 1234, gas: 25.500000  
[1] num: 4567, gas: 57.700000  
[2] num: 7890, gas: 28.700000
```

【課題11-1】

ファイル入出力バージョンの作成

main()の処理に、**load_cars()**と**save_cars()**を付け加えて動作を確認して下さい。「実行するたびに、メンバgasの値が増え続ける」プログラムができあがります。（以下参照）

[mainでの処理(ファイル入出力バージョン)]

```
Car cars3[3];          /* ←初期値を削除 */  
load_cars(cars3);      /* ←追加 (ファイルから読み込む処理) */  
add_gas(cars3, 1, 5.5);  
add_gas(cars3, 2, 8.2);  
show_cars(cars3, "add_gas: cars3");  
save_cars(cars3);      /* ←追加 (ファイルへ書き込む処理) */
```

【課題1 1-1】

[実行結果]

```
$ ./a.out
(loading)
--- add_gas: cars3 ---
[0] num: 1234, gas: 25.500000
[1] num: 4567, gas: 63.200000      (←57.7に5.5が加算された)
[2] num: 7890, gas: 36.900000      (←28.7に8.2が加算された)
(saving)

$ ./a.out
(loading)
--- add_gas: cars3 ---
[0] num: 1234, gas: 25.500000
[1] num: 4567, gas: 68.700000      (←63.2に5.5が加算された)
[2] num: 7890, gas: 45.100000      (←36.9に8.2が加算された)
(saving)
```

【課題11-2】

構造体型struct Carの配列に対して、「全ての要素のメンバgasを加算する」関数add_gas_array()を作成して下さい。

[この関数のプロトタイプ宣言]

```
void add_gas_array(Car *p, float g);
```

```
/* 引数のポインタpが参照している配列の全要素に対して、  
   メンバgasに引数gを加算する */
```

```
/* add_gas()で作った加算処理を、for文を使って全要素に対して  
   加算するように作る */
```

【課題11-2】

[mainでの処理]

```
Car cars4[3] = {  
    {1234, 10.4},  
    {4567, 33.2},  
    {7890, 25.6}  
};  
Car cars5[3] = {  
    {1357, 20.5},  
    {3579, 15.0},  
    {5791, 17.3}  
};  
add_gas_array(cars4, 1.5);  
add_gas_array(cars5, 0.5);  
show_cars(cars4, "add_gas_array: cars4");  
show_cars(cars5, "add_gas_array: cars5");
```

【課題11-2】

[実行結果]

```
--- add_gas_array: cars4 ---
```

```
[0] num: 1234, gas: 11.900000
```

(←全要素のメンバgasに1.5が加算された)

```
[1] num: 4567, gas: 34.700000
```

```
[2] num: 7890, gas: 27.100000
```

```
--- add_gas_array: cars5 ---
```

```
[0] num: 1357, gas: 21.000000
```

(←全要素のメンバgasに0.5が加算された)

```
[1] num: 3579, gas: 15.500000
```

```
[2] num: 5791, gas: 17.800000
```


【課題11-2】

ファイル入出力バージョンの作成

main()の処理に、load_cars()とsave_cars()を付け加えて動作を確認して下さい。「実行するたびに、全要素のメンバgasの値が増え続ける」プログラムができあがります。（以下参照）

まず、「cars.csv」ファイルの中身（つまり配列の初期値）を次のように変更する

1357,20.5
3579,15.0
5791,17.3

【課題11-2】

[mainでの処理(ファイル入出力バージョン)]

/* 以下からcars4の記述を削除 */

Car cars5[3]; /* ←初期値を削除 */

load_cars(cars5); /* ←追加 (ファイルから読み込む処理) */

add_gas_array(cars5, 0.5);

show_cars(cars5, "add_gas_array: cars5");

save_cars(cars5); /* ←追加 (ファイルへ書き込む処理) */

【課題11-2】

[実行結果]

```
$ ./a.out
(loading)
--- add_gas_array: cars5 ---
[0] num: 1357, gas: 21.000000    (←20.5に0.5が加算された)
[1] num: 3579, gas: 15.500000    (←15.0に0.5が加算された)
[2] num: 5791, gas: 17.800000    (←17.3に0.5が加算された)
(saving)

$ ./a.out
(loading)
--- add_gas_array: cars5 ---
[0] num: 1357, gas: 21.500000    (←上のgasからさらに0.5が加算された)
[1] num: 3579, gas: 16.000000
[2] num: 5791, gas: 18.300000
(saving)
```

【課題11-3】

構造体型struct Carの配列に対して、「指定した要素を、引数で与えた構造体に置き換える」関数set_car()を作成して下さい。

[この関数のプロトタイプ宣言]

```
void set_car(Car *p, int i, Car *car);
```

```
/* 仮引数のポインタpが参照している配列のi番目の要素に、  
   ポインタcarが参照している構造体を代入する */
```

```
/* 「pが参照している配列のi番目の要素」は*(p+i)で表す */
```

【課題11-3】

[mainでの処理]

```
Car cars6[3] = {  
    {1234, 25.5},  
    {4567, 52.2},  
    {7890, 20.5}  
};  
Car car1 = {3333, 10.5};  
Car car2 = {5555, 30.5};  
set_car(cars6, 0, &car1);  
set_car(cars6, 2, &car2);  
show_cars(cars6, "set_car: cars6");
```

[実行結果]

```
--- set_car: cars6 ---  
[0] num: 3333, gas: 10.500000  
[1] num: 4567, gas: 52.200000  
[2] num: 5555, gas: 30.500000
```

【課題11-4】

第10週 課題10-4で作成した構造体型struct Time
に対して、以下の関数を作成して下さい。

- ▶ 「構造体の配列の全ての要素を出力する」関数show()
- ▶ 「構造体の配列をCSVファイルに保存する」関数save()
- ▶ 「構造体の配列をCSVファイルから読み込む」関数load()

[この関数のプロトタイプ宣言]

```
void show(Time *p, char *str);  
    /* 関数show_cars()を参考に、構造体型struct Timeに対応させる */  
void save(Time *p);  
    /* 関数save_cars()を参考に、構造体型struct Timeに対応させる */  
void load(Time *p);  
    /* 関数load_cars()を参考に、構造体型struct Timeに対応させる */
```


【課題11-4】

[mainでの処理]

```
Time times1[5] = {
    {12, 30},
    {10, 20},
    {14, 40},
    {13, 35},
    {11, 25}
};
Time times2[5];
show(times1, "times1");
save(times1);    /* times1の内容を保存 */
load(times2);    /* 保存した内容をtimes2に読込 */
show(times2, "times2");
```

【課題11-4】

[実行結果]

```
--- times1 ---
```

```
[0] 12:30
```

```
[1] 10:20
```

```
[2] 14:40
```

```
[3] 13:35
```

```
[4] 11:25
```

```
(saving)
```

```
(loading)
```

```
--- times2 ---
```

```
[0] 12:30
```

```
[1] 10:20
```

```
[2] 14:40
```

```
[3] 13:35
```

```
[4] 11:25
```

小テストについて

小テストの注意点

- 他人の力は借りずに、自分だけでプログラムを作成する。（つまり定期試験と同様）
- 小テスト中は、演習室外へのネットワークアクセスは遮断される。

小テストについて

小テスト中に参照できるもの

- 教科書, 配付資料
- 自分のホームディレクトリ（ホームフォルダ）以下に保存されているファイル
- 小テストでは紙媒体のものは参照可能
- 上記以外の情報を参照することは不正行為とする
例：USBで接続された機器に保存されているファイルの参照
ネットワークを介した情報の参照、など