

プログラミング基礎

<http://bit.ly/prog2d>


構造体のリスト

後期 第12週

2017/12/18

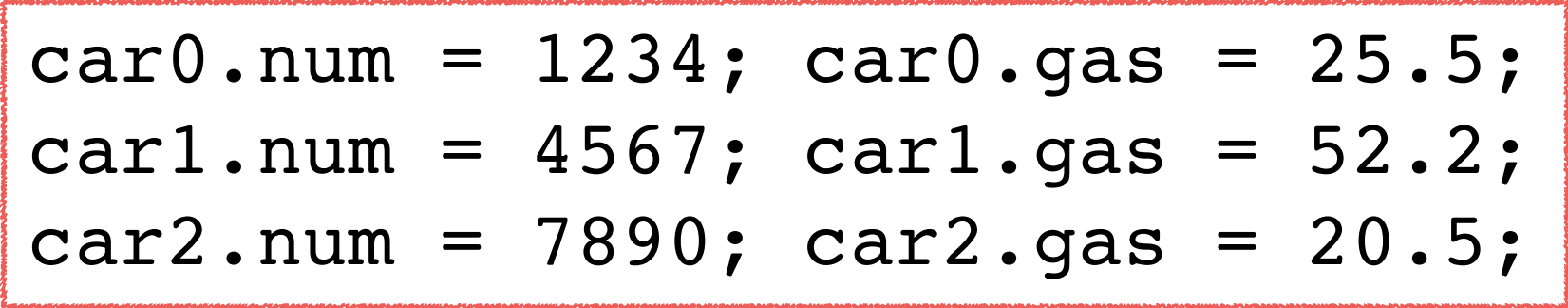
【Point 1】 「struct Car *」 という型のnextは、自身と同じ構造体型を参照できるポインタとなる (p.374)

```
1: #include <stdio.h>
2:
3: typedef struct Car {
4:     int num;
5:     double gas;
6:     struct Car *next;
7: } Car;
8:
9: void show_carlist(Car *start, char *str);
10:
```

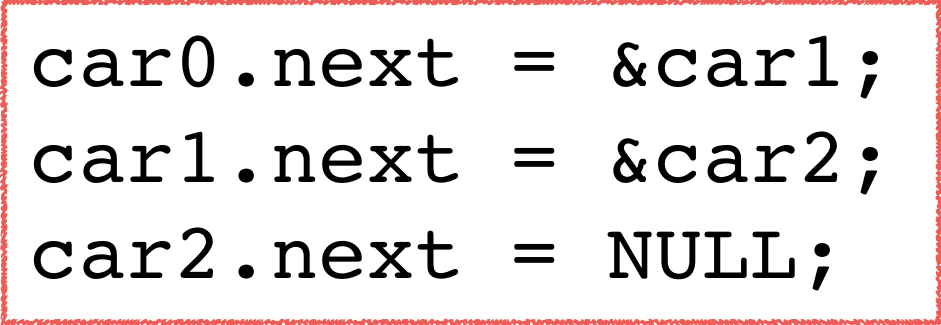


```
11: int main(void)
12: {
13:     Car car0, car1, car2, car3, car4;
14:     Car *pcar;
15:
16:     car0.num = 1234; car0.gas = 25.5;
17:     car1.num = 4567; car1.gas = 52.2;
18:     car2.num = 7890; car2.gas = 20.5;
19:
20:     car0.next = &car1;
21:     car1.next = &car2;
22:     car2.next = NULL;
```

3個の構造体のメンバに値を代入する



```
car0.num = 1234; car0.gas = 25.5;
car1.num = 4567; car1.gas = 52.2;
car2.num = 7890; car2.gas = 20.5;
```



```
car0.next = &car1;
car1.next = &car2;
car2.next = NULL;
```

【Point 1】

- car0のメンバnextがcar1を参照する (car0の次にcar1がつながる)
- car1の次にcar2がつながる
- car2のメンバnextをNULLとする (リストの末尾とする)

【Point 2】 NULLになるまで繰り返す

【Point 2】 car0から繰り返しを開始する

```
23:   for(pcar = &car0; pcar!=NULL;  
      pcar = pcar->next) {
```

【Point 2】 nextをたどって次の要素を参照する

```
24:       printf("num: %d, gas: %lf\n",  
                pcar->num, pcar->gas);
```

```
25:   }
```

```
26:   show_carlist(&car0, "car list");
```

```
27:   show_carlist(&car1, "from car1");
```

```
28:
```

```
29:   return 0;
```

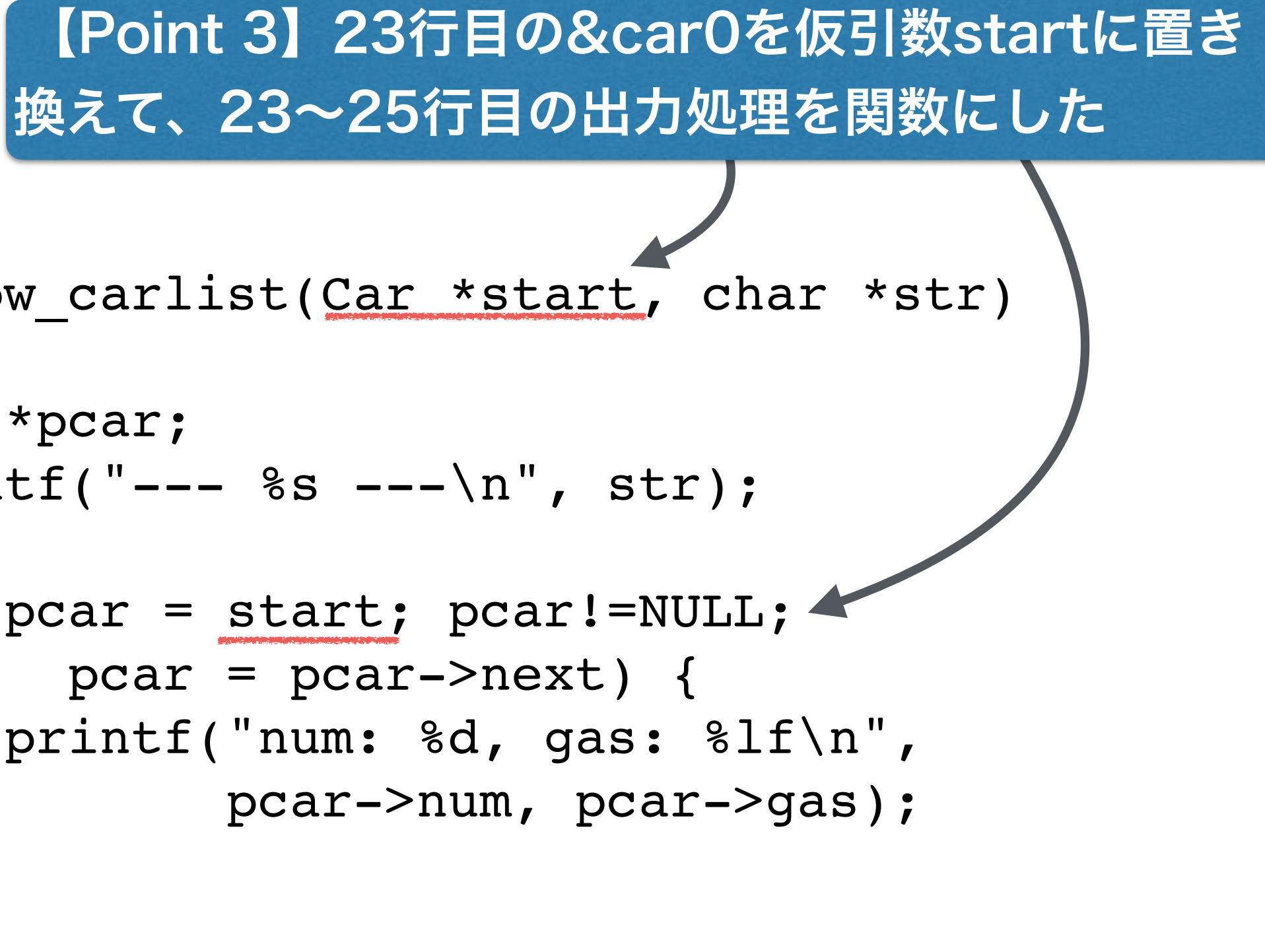
```
30: }
```

```
31:
```

【Point 3】 23行目の&car0を仮引数startに置き換えて、23～25行目の出力処理を関数にした

```
32: void show_carlist(Car *start, char *str)
33: {
34:     Car *pcar;
35:     printf("--- %s ---\n", str);

36:     for(pcar = start; pcar!=NULL;
           pcar = pcar->next) {
37:         printf("num: %d, gas: %lf\n",
                 pcar->num, pcar->gas);
38:     }
39: }
```

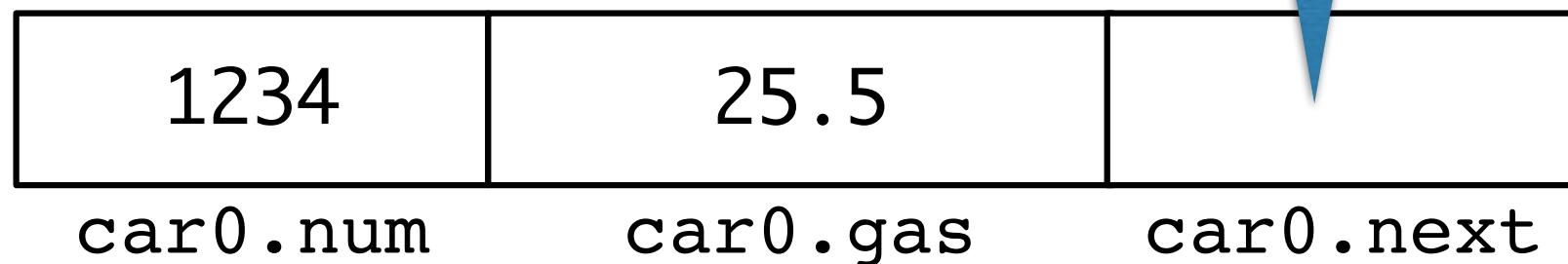


【Point 1】の補足

前回作成した構造体型struct Carに、「**struct Carを参照するためのポインタ**」であるメンバnextを追加しています

変数car0のイメージ

struct Car型（Car型）の構造体を参照するためのポインタnextがメンバとなる



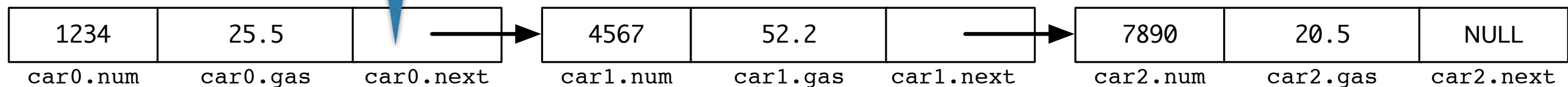
※ 16行目時点での様子

【Point 1】 の補足

この構造体を使うと、メンバ`next`で一列につながったリストの構造（線形リスト）を作ることができます。（p.377 図11-11）

20～22行目の代入処理後のイメージ

このポインタに`car1`の先頭アドレスが代入されている
(つまり、ポインタ`car0.next`が`car1`を参照している)



このポインタに`NULL`が代入されている
(つまり、リストの末尾になる)

【Point 2】の補足

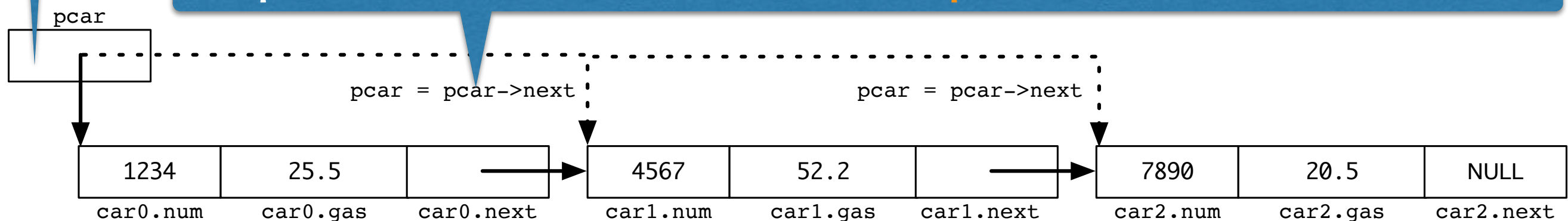
リストに対して繰り返して処理をする場合は、

「`pcar = pcar->next`」のように、**nextの参照をたどっていきます。**

23～25行目の繰り返し処理の様子

最初はcar0を参照している

pcarがcar0を参照している場合、「`pcar->next`」は「**car0のメンバnextに格納されているアドレス**（つまりcar1のアドレス）」となり、このアドレスをpcarに代入しているので、その結果、**pcarはcar1を参照する**



「`pcar=pcar->next`」で、**このNULLがpcarに代入されると繰り返し処理が終了**

【練習12-1】

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。

【練習12-2】

リストの末尾にさらにCarの要素を追加して、実行結果を確認してみましょう。

サンプルプログラムのmainの最後（28行目）に以下を追加する

```
car3.num = 2468; car3.gas = 10.5;    /* car3のメンバに値を入れる */  
car2.next = &car3;                  /* car2の次にcar3をつなぐ */  
car3.next = NULL;                   /* car3をリストを末尾とする */  
show_carlist(&car0, "car list");    /* car0～car3が出力される */
```

【課題12-1】

関数show_carlist()を参考に、「リストの要素のメンバnumが**仮引数gより大きい場合**、車の情報を表示する」関数show_greater()を作成して下さい。

[この関数のプロトタイプ宣言]

```
void show_greater(Car *start, int g);
```

```
/* show_carlist()の処理を基に作れる */
```

```
/* 「pcarが参照しているメンバnumが、仮引数gよりも大きい場合」に  
出力するという条件分岐を追加する */
```

【課題12-1】

[mainでの処理（練習12-2の後に追加したとする）]

```
show_greater(&car0, 3000);    /* car0～car3に対して処理する */  
car4.num = 3579; car4.gas = 15.5;  
car3.next = &car4;          /* car3の次にcar4をつなぐ */  
car4.next = NULL;  
show_greater(&car0, 3000);    /* car0～car4に対して処理する */
```

[実行結果]

```
--- show_greater() ---  
num: 4567, gas: 52.200000  
num: 7890, gas: 20.500000  
--- show_greater() ---  
num: 4567, gas: 52.200000  
num: 7890, gas: 20.500000  
num: 3579, gas: 15.500000
```

【課題12-2】

「リストの全ての要素のメンバgasの平均値を求めて返す」関数average_gas()を作成して下さい。

[この関数のプロトタイプ宣言]

```
double average_gas(Car *start);
```

```
/* リストの全要素に対する繰り返し処理の中で以下の処理をする */
```

```
/* ・要素の個数を数える */
```

```
/* ・各要素のメンバgasの合計を求める */
```

```
/* 繰り返し処理後に平均値を求め、関数の戻り値とする */
```

【課題12-2】

[mainでの処理（課題12-1の後に追加したとする）]

```
show_carlist(&car0, "car list");  
printf("car0以降の平均値: %lf\n", average_gas(&car0));  
printf("car3以降の平均値: %lf\n", average_gas(&car3));
```

[実行結果]

```
--- car list ---  
num: 1234, gas: 25.500000  
num: 4567, gas: 52.200000  
num: 7890, gas: 20.500000  
num: 2468, gas: 10.500000  
num: 3579, gas: 15.500000  
car0以降の平均値: 24.840000  
car3以降の平均値: 13.000000
```


【課題12-3】

「リストの全ての要素のメンバgasの値を、先頭（つまりstartが参照している）の要素のメンバgasに移す」関数move_gas()を作成して下さい。

[この関数のプロトタイプ宣言]

```
void move_gas(Car *start);
```

```
/* リストの全要素に対する繰り返し処理の中で以下の処理をする */
```

```
/* pcarがstartと等しくない場合... */
```

```
/* startが参照しているgasへ、pcarが参照しているgasを加算する */
```

```
/* 加算したpcarのgasは0にする */
```

【課題12-3】

[mainでの処理（課題11-1の後に追加したとする）]

```
move_gas(&car2);          /* car2へ、car3～car4のgasを移動する */  
show_carlist(&car0, "moved car list");  
move_gas(&car0);          /* car0へ、car1～car4のgasを移動する */  
show_carlist(&car0, "moved car list");
```

[実行結果]

```
--- moved car list ---  
num: 1234, gas: 25.500000  
num: 4567, gas: 52.200000  
num: 7890, gas: 46.500000  
num: 2468, gas: 0.000000  
num: 3579, gas: 0.000000  
--- moved car list ---  
num: 1234, gas: 124.200000  
num: 4567, gas: 0.000000  
num: 7890, gas: 0.000000  
num: 2468, gas: 0.000000  
num: 3579, gas: 0.000000
```

【課題12-4】

構造体型struct Timeに対して、線形リストとなるように、次のようにメンバnextを追加します。

```
typedef struct Time {  
    int hour;  
    int minute;  
    struct Time *next;    /* メンバnextを追加 */  
} Time;
```

課題は次のスライドに続きます

【課題12-4】

この構造体で作られた線形リストに対して、「各要素の時間の情報を出力する」関数show_timelist()を作成して下さい。

[この関数のプロトタイプ宣言]

```
void show_timelist(Time *start, char *str);
```

```
/* 関数show_carlist()を参考に、struct Timeに対応させる  
   (出力方法は第9回show_time()と同じ) */
```

【課題12-4】

[mainでの処理]

```
Time time0 = {12, 30, NULL};    /* リストの要素を3つ作る */
Time time1 = {10, 20, NULL};
Time time2 = {14, 50, NULL};
time0.next = &time1;             /* 要素をつないでリストにする */
time1.next = &time2;
show_timelist(&time0, "time list");
show_timelist(&time1, "from time1");
```

[実行結果]

```
--- time list ---
12:30
10:20
14:50
--- from time1 ---
10:20
14:50
```

まだ余裕のある人は…【課題12-5】

線形リストは、メンバnextの参照を変更すれば、
簡単にリストの順番を並び替えることができます。

課題11-1以降のプログラムを、car0～car4の順の
リストではなく、car4～car0と逆順につなぐように
main()の処理を変更し、show_carlist()によって逆
順で出力されることを確認して下さい。

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