

プログラミングII

<http://bit.ly/Prog3i>

メモリの動的確保と構造体

前期 第5週

2019/5/14

今回は

プログラムの実行中に、**構造体**に対して
必要な分だけメモリを確保する方法
について説明します。

どんな時に使うのか

▶ 構造体がリスト（線形リスト）の構造

- 配列とは異なり、プログラム記述時は要素数を指定しない
- プログラム実行中に、リストへ新しい要素を追加/削除する
- 新しい要素を作る際に、構造体の動的メモリ確保を行う（関数mallocを呼び出す）

▶ まずは、構造体を使ったリスト構造について復習

【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

【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) {  
  
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 2】nextをたどって次の要素を参照する

【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);

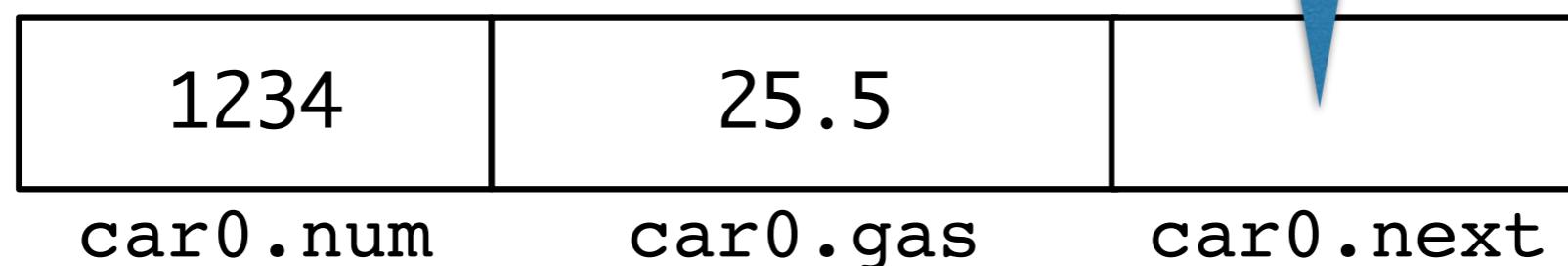
36:     for(pcar = start; pcar!=NULL;
37:         pcar = pcar->next) {
38:         printf("num: %d, gas: %lf\n",
39:                pcar->num, pcar->gas);
}
```

【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



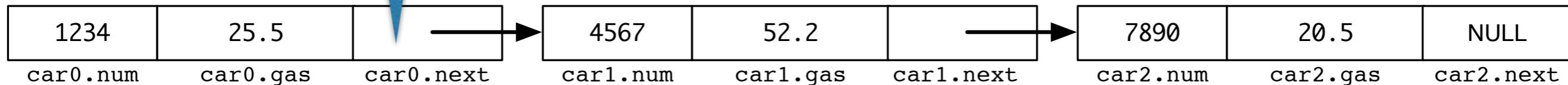
※ 16行目時点での様子

【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)

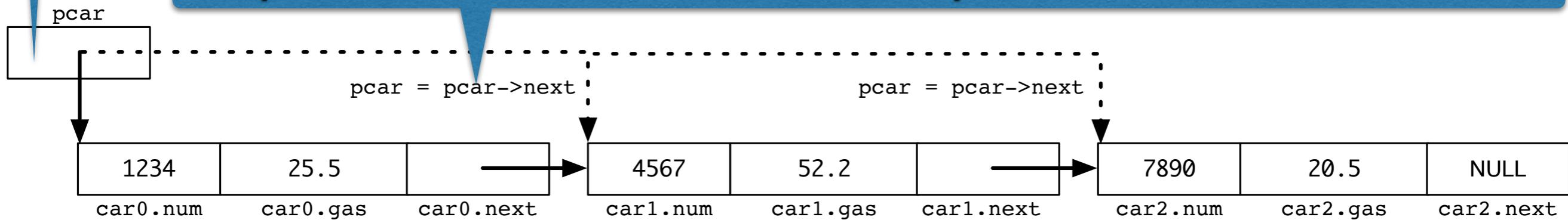
【Point 2】の補足

リストに対して繰り返して処理をする場合は、
「`pchar = 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

pcharがcar0を参照している場合、「`pchar->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pcharに代入しているので、その結果、**pcharはcar1を参照する**



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

リストに新しい要素を追加できるようにする

以下のような処理をするプログラムを作ります

- ▶ リストの先頭を表す構造体headを用意
- ▶ 関数malloc()を使って新しい要素のメモリを確保
- ▶ 何回でも使えるように、メモリ確保の処理を関数add_car()として定義する

関数malloc() を使うためにインクルードする

```
1: #include <stdio.h>
2: #include <stdlib.h>
3:
4: typedef struct Car {
5:     int num;
6:     double gas;
7:     struct Car *next;
8: } Car;
9:
10: void show_carlist(Car *start, char *str);
11: void add_car(Car *p);
12:
```

リストを作るための
構造型の宣言

```
13: int main(void)  
14: {  
15:     Car head;  
16:     Car *new;  
17:  
18:     head.num = 0; head.gas = 0;  
19:     head.next = NULL;  
20:  
21:     show_carlist(&head, "head (1)");  
22:  
23:     new = (Car *)malloc(sizeof(Car));  
24:     new->num = 1234; new->gas = 25.5;
```

リストの先頭要素となる構造体変数

新しく作る要素を参照するポインタ

【Point 1】先頭要素のみのリストを作る

【Point 1】構造体型Carの要素をメモリに確保し、
ポインタnewがその領域を参照する

【Point 1】newが参照しているメンバ
に値をそれぞれ代入する

【Point 2】新しく作った要素 (newが参照している要素) のメンバnextが、head.nextと同じ場所を参照先する (ここではNULLが代入される)

【Point 2】head.nextが新しく作った要素を参照する
(つまり、headの後に新しく作った要素がつながる)

```
25:     new->next = head.next;
```

```
26:     head.next = new; ←
```

```
27:
```

```
28:     show_carlist(&head, "head (2)");
```

```
29:
```

```
30:     add_car(&head);
```

```
31:     show_carlist(&head, "head (3)");
```

```
32:     add_car(&head);
```

```
33:     show_carlist(&head, "head (4)");
```

```
34:
```

```
35:     return 0;
```

```
36: }
```

【Point 3】さらに要素
をリストに追加してみる

この関数は前回と全く同じ

```
37:  
38: void show_carlist(Car *start, char *str)  
39: {  
    /* この関数の中身は前回と同じ */  
45: }  
46:  
47: void add_car(Car *p) ←  
48: {  
    Car *new;  
50:  
51:     new = (Car *)malloc(sizeof(Car));  
52:     new->num = 1111; new->gas = 11.1;  
53:     new->next = p->next; ←  
54:     p->next = new; ←  
55: }
```

headのアドレスを仮引数pへ参照渡している

25, 26行目の処理を、headから仮引数pに置き換えた

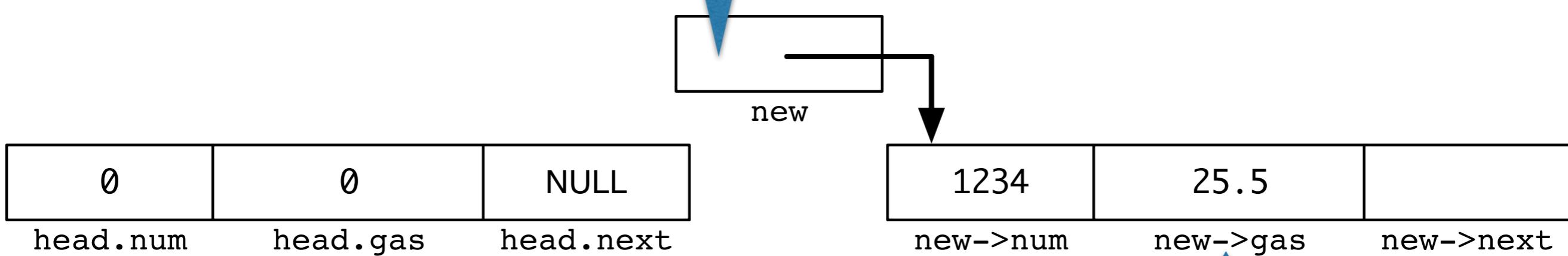
23, 24行目の処理を関数内に入れた（区別するため、メンバに代入する値は変えてある）

【Point 1】の補足

malloc()を使って構造体型Carの領域をメモリに確保し、ポインタnewを使ってその領域を参照します。

この処理のイメージ

ポインタnewが新しく作った領域を参照する



headは構造体型の変数なので、
ドット演算子でメンバにアクセスする

アロー演算子を使って各メンバ
にアクセスできる

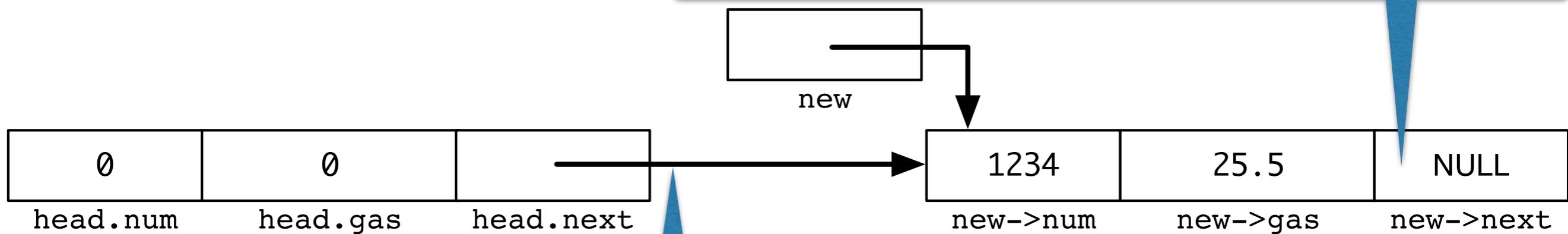
※ 24行目時点での様子

【Point 2】の補足

メンバnextの参照先を変更することで、結果的に
headの後ろに新しい要素が追加されます。

この処理のイメージ

25行目の「`new->next = head.next`」
によって `NULL` が代入される



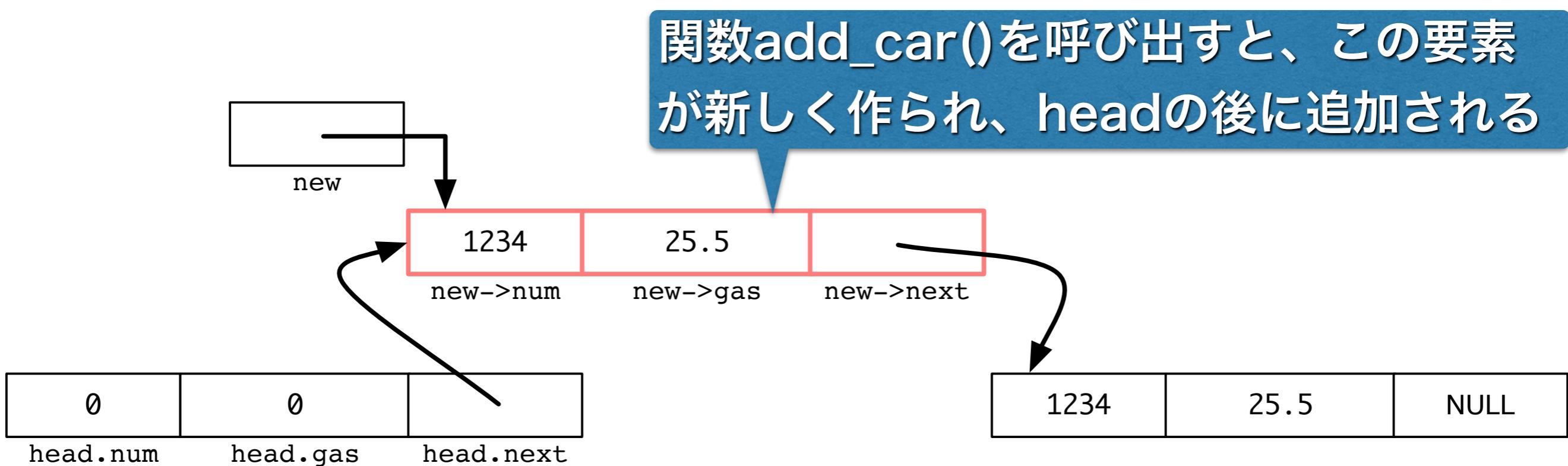
26行目の「`head.next = new`」
によって新しく作られた要素を参照する

※ 26行目時点での様子

【Point 3】の補足

関数add_car()を呼び出す度に、headの後にリストの要素を追加できるようになります。

この処理のイメージ



※ 30行目の関数呼び出し後の様子

【課題の準備】

演習室で作業する前に、以下のコマンドを入れるだけで準備が完了する

```
$ mygitclone 「自分のGitHubユーザ名」  
$ cd prog3i-(ユーザ名)  
$ ./myconf
```

※本体をシャットダウンするまでは、
上記「mygitclone」と「myconf」の設定は有効です

【課題の準備】

以下の流れで、課題のプログラムを作るためのフォルダを準備しましょう。

1. 端末を起動して、以下のコマンドを実行して前期第5週のフォルダを作る

\$ cd prog3i-(ユーザ名) (←既に移動しているなら不要)

\$ mkdir week105

\$ cd week105

※課題で作るファイル名は各自で決めて構いません。
(全ての課題を1つのファイルでまとめて作っても良い)

【練習5-1】

線形リストのサンプルプログラムsample105-1.cを
コンパイルして、実行結果を確認してみましょう。
(配付資料と同じ場所からダウンロード可能)

【練習5-2】

線形リストに新しい要素を追加するサンプルプログラムsample105-2.cをコンパイルして、実行結果を確認してみましょう。

(配付資料と同じ場所からダウンロード可能)

【課題5-1】

関数add_car()をもとに、「引数を使って、新しく作成する要素のナンバーとガソリンの量を指定できる」
関数add_car2()を作成してください。

[この関数のプロトタイプ宣言]

```
void add_car2(Car *p, int n, double g);  
  
/* 領域を確保した要素のメンバnumに仮引数n、  
   gasに仮引数gを代入するように変更する */
```

【課題5-1】

[mainでの処理]

```
Car head2;
head2.num = 0; head2.gas = 0; head2.next = NULL;
show_carlist(&head2, "head2 (1)");
add_car2(&head2, 1357, 40.3);
add_car2(&head2, 2468, 33.8);
add_car2(&head2, 3579, 26.1);
show_carlist(&head2, "head2 (2)");
```

[実行結果]

```
--- head2 (1) ---
num: 0, gas: 0.000000
--- head2 (2) ---
num: 0, gas: 0.000000
num: 3579, gas: 26.100000
num: 2468, gas: 33.800000
num: 1357, gas: 40.300000
```

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題5-1提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

【課題5-2】

課題5-1の関数add_car2()をもとに、「引数で与えられた車のナンバーが奇数の場合のみ新しい要素を追加し、与えられたガソリンの量が0ならば10とする」関数add_car3()を作成してください。

[この関数のプロトタイプ宣言]

```
void add_car3(Car *p, int n, double g);
```

/* 課題5-1で作成した関数において、以下のような条件を追加する */

/* ● nが奇数の場合のみ、malloc()で領域を確保する */

/* ● (上記の条件を満たした場合、)

gが0の場合、作成した要素のメンバgasに10を代入し、

0でない場合、gasにgを代入する */

【課題5-2】

[mainでの処理]

```
Car head3;
head3.num = 0; head3.gas = 0; head3.next = NULL;
show_carlist(&head3, "head3 (1)");
add_car3(&head3, 1357, 40.3);
add_car3(&head3, 2468, 33.8); /* ナンバーが偶数なので追加されない */
add_car3(&head3, 3579, 0);    /* ガスの量が0なので10として追加される */
show_carlist(&head3, "head3 (2)");
```

[実行結果]

```
--- head3 (1) ---
num: 0, gas: 0.000000
--- head3 (2) ---
num: 0, gas: 0.000000
num: 3579, gas: 10.000000
num: 1357, gas: 40.300000
```

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題5-2提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

【課題5-3】

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5-3】

[mainでの処理]

```
Car head4;
head4.num = 0; head4.gas = 0; head4.next = NULL;
add_car2(&head4, 1357, 40.3);
add_car2(&head4, 2468, 33.8);
show_greater(&head4, 2467);      /* 2つの要素で出力してみる */
add_car2(&head4, 3579, 26.1);   /* 新しい要素を追加して... */
show_greater(&head4, 2467);      /* 3つの要素で出力してみる */
```

[実行結果]

```
--- show_greater() ---
num: 2468, gas: 33.800000
--- show_greater() ---
num: 3579, gas: 26.100000
num: 2468, gas: 33.800000
```

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題5-3提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

【課題5-4】

「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5-4】

[mainでの処理]

```
Car head5;
head5.num = 0; head5.gas = 0; head5.next = NULL;
add_car2(&head5, 1357, 40.3);
add_car2(&head5, 2468, 33.8);
printf("平均値(1): %lf\n", average_gas(&head5));
add_car2(&head5, 3579, 26.1);
printf("平均値(2): %lf\n", average_gas(&head5));
```

[実行結果]

```
平均値(1): 24.700000
平均値(2): 25.050000
```

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題5-4提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

次回は

- ▶ コマンドライン引数について
- ▶ 小テスト（メモリの動的確保の内容）
- ▶ 前期中間試験について