プログラミング基礎

http://bit.ly/prog2d

構造体のリストに対する操作

後期 第13週

2018/1/15

関数malloc() を使うためにインクルードする

リストを作るための
構造体型の宣言

```
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:
```

```
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

```
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: {
49:     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: }
```

この関数は前回と全く同じ

headのアドレスを仮引数pへ参照渡している

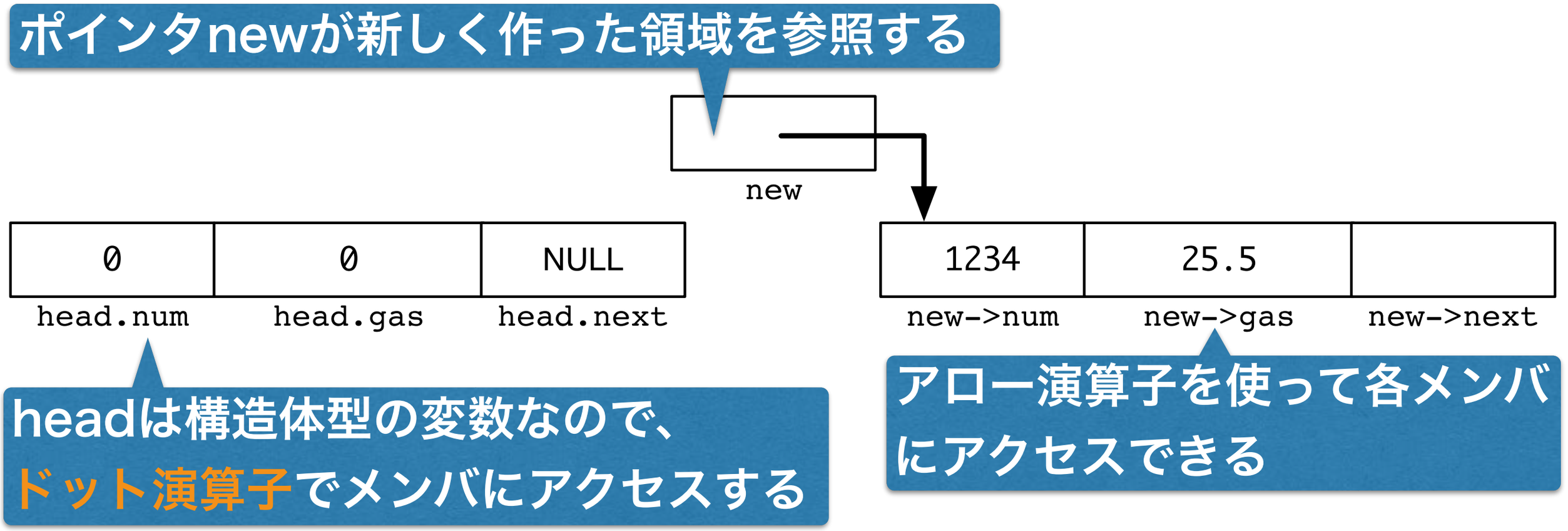
25, 26行目の処理を、headから仮引数pに置き換えた

23, 24行目の処理を関数内に入れた（区別するために、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

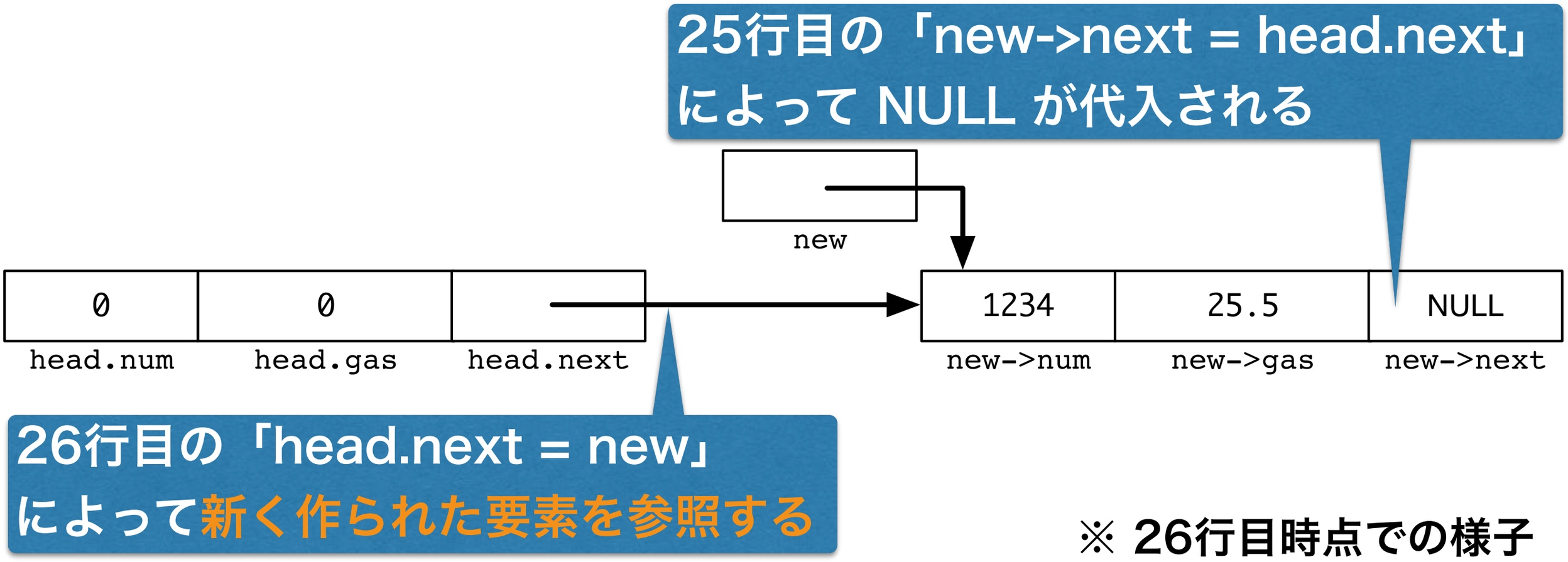


※ 24行目時点での様子

【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

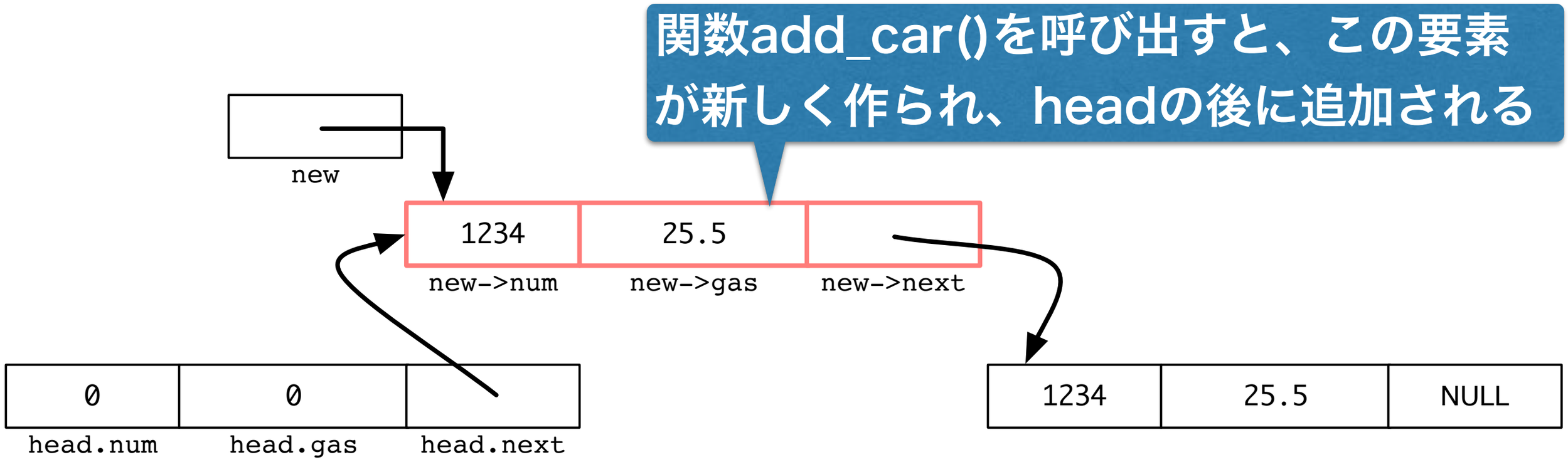


※ 26行目時点での様子

【Point 3】の補足

関数add_car()を呼び出す度に、headの後ろ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行目の関数呼出し後の様子

コマンドライン引数について

プログラム実行時（つまり「a.out」を実行する時）に、ユーザがプログラムに与えることができる引数

これまでmain()で使っていた仮引数voidの代わりに、
仮引数argcとargvを使う (p.421～)

```
1: #include <stdio.h>
2:
3: int main(int argc, char *argv[])
4: {
5:     int i;
6:
7:     printf("コマンドライン引数の個数: %d\n", argc);
8:     for(i=0; i<argc; i++) {
9:         printf("argv[%d]: %s\n", i, argv[i]);
10:    }
11:
12:    return 0;
13: }
```

argvが文字列の配列となり、argcが配列の要素数となって、プログラムが開始する

コマンドライン引数の例

「./a.out」の後に、文字列を空白で区切って指定する

端末（ターミナル）

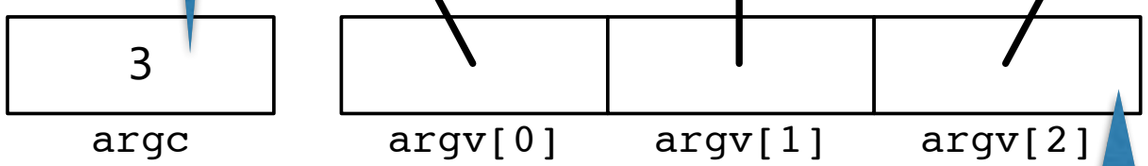
```
$   ./a.out    myfile1.txt    myfile2.txt
```

実行

argcは配列の要素数が入っている

プログラム

"./a.out" "myfile1.txt" "myfile2.txt"



「./a.out」も含めて、argvがこれらの文字列を参照している

```
int main(int argc, char *argv[])
{
    ...
}
```

【練習13-1】

線形リストの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ょう。

【練習13-2】

コマンドライン引数の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ょう。

実行する際は、「./a.out」の後に文字列をいくつか指定してみましょう。

【課題13-1】

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13-1】

[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13-2】

課題12-1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13-1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13-2】

[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13-3】

第9回に示した「ファイルを1文字ずつコピーするプログラム」に対して、「コピー元とコピー先のファイル名をコマンドライン引数で指定できる」ように変更してください。（以下参考）

- **fopen()で開くファイル名の箇所を、argvの文字列を使うように変更する（1番目がコピー元のファイル名、2番目がコピー先のファイル名となっている）**
- **argcが3ではない場合はプログラムを終了するようにする**

【課題13-3】

[実行時の様子]

```
$   ./a.out   testIn.txt   a.txt        (testIn.txtがa.txtにコピーされる)
$   ./a.out   a.txt   b.txt             (a.txtがb.txtにコピーされる)
$   ./a.out   a.txt   b.txt c.txt       (引数の個数が3ではない場合)
```

パラメータの数が違います。

【課題13-4】

コマンドライン引数で指定した文字列に対して、「それぞれの文字列の長さ（つまり、文字数）とその合計を求める」プログラムを作成して下さい。

- argvの文字列に対して、標準ライブラリ関数strlen()を使って文字列の長さを求める
- 上記の長さを求める処理をargvの要素数分繰り返す処理を作る
- 出力する情報は、以下の実行結果を参照

【課題13-4】

[実行結果]

```
$  ./a.out  pointer  malloc  free  stream  struct
pointer(7)
malloc(6)
free(4)
stream(6)
struct(6)
result: 29
```

まだ余裕のある人は…【課題13-5】

コマンドライン引数で指定した文字列に対して、「それぞれの文字列を整数に変換してその合計を求める」プログラムを作成して下さい。

- argvの文字列に対して、標準ライブラリ関数 atoi() を使って整数に変換する（p.465参照）
- 出力する情報は、以下の実行結果を参照

【課題13-5】

[実行結果]

```
$   ./a.out   8   3                    (2個の整数を指定した場合)
8 + 3 = 11

$   ./a.out   3   1   4   2   8   6    (6個の整数を指定した場合)
3 + 1 + 4 + 2 + 8 + 6 = 24
```

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

小テストの追試について

1月22日（月）授業終了後に、小テストの追試験を実施します。未受験の小テストがある人は、この時に必ずまとめて受けて下さい。
（やむを得ず欠席する際は事前に連絡を下さい。）

未受験分の小テストは0点として評価します。

今回実施した小テスト

- **第11週分（構造体の配列）**
- **第12週分（構造体のリスト）**
- **第13週分（構造体のリストに対する操作）**