プログラミング基礎

http://bit.ly/prog2d

構造体の基本とポインタ

後期 第10週

2017/12/4

構造体とは

異なる型の値（変数）をまとめて

新しい型を作る機能

構造体型

以降のp.350～p.362のプログラムと同等

```
1:   #include <stdio.h>
2:
3:   struct Car {
4:       int num;
5:       double gas;
6:   };
7:
```

構造体型の宣言 (p.350)

num（車のナンバー）という名前のint型のメンバ

gas（ガソリンの量）という名前のdouble型のメンバ

「struct」というキーワードを使って、新しい構造体型を宣言し、structの括弧内に、構造体型に含めたい変数（メンバと呼びます）を宣言します。

```
 8:   int main(void)
 9:   {
10:       struct Car car1, car2;
11:       struct Car car3 = {1357, 20.3};
12:
```

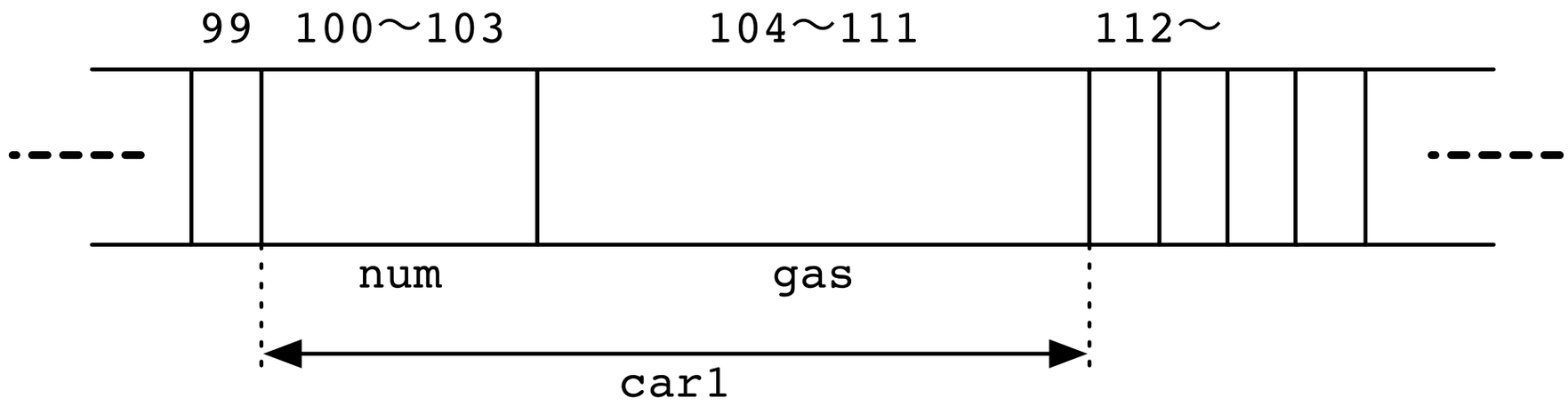
構造体変数を宣言する (p.352)

構造体変数を宣言し、同時に初期化する (p.358)

{ }内の値の順番と、構造体で宣言したメンバの順番は対応している

構造体型を型として宣言した変数を構造体変数といいます。

この例では、「struct Car」という構造体型に対して、car1とcar2という構造体変数を宣言しています。構造体変数car1は、メモリ上では、メンバnumとgasの格納場所が連続してメモリ上に確保されます。

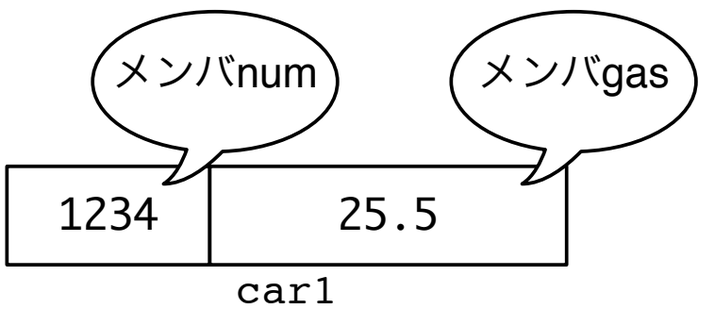


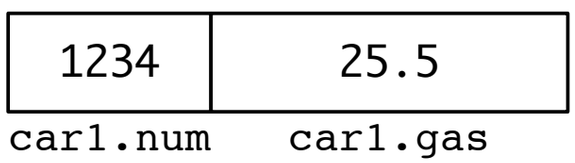
構造体変数car1のメンバnumに1234を代入 (p.353)
ドット演算子を使ってメンバにアクセスする

```
13:         car1.num = 1234;
14:         car1.gas = 25.5;
15:         car2.num = 5678;
16:         car2.gas = 10.5;
17:
```

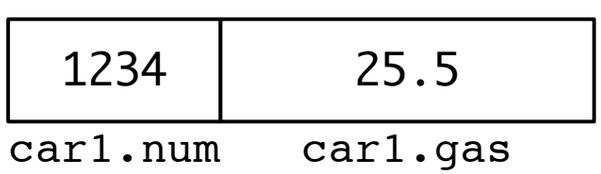
構造体変数のメンバにアクセスする場合は、構造体変数とメンバをドットで区切りって表します。（ドット演算子といいます）

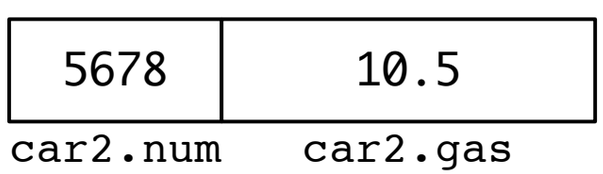
構造体変数car1は次のように表します。（授業では下図右のように表す）

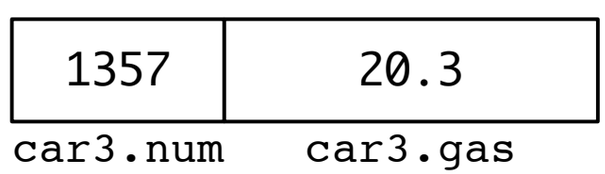




この時点で、car1, car2, car3は次のようになります。







構造体変数のメンバに格納されている値を使う時もドット演算を使う
（メンバnumはint型なので%dを指定している）

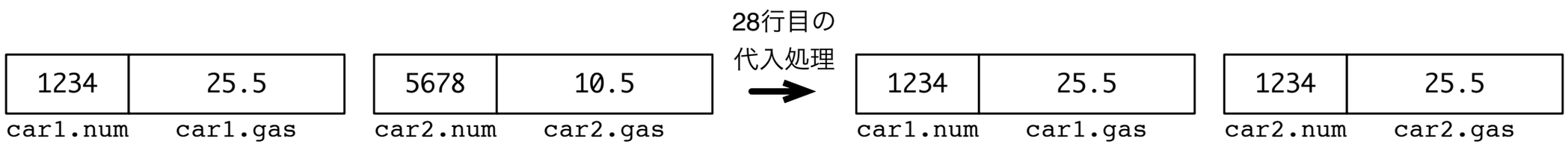
```
18:         printf("(car1) num: %d, gas: %f\n",
                    car1.num, car1.gas);
19:         printf("(car2) num: %d, gas: %f\n",
                    car2.num, car2.gas);
20:         printf("(car3) num: %d, gas: %f\n",
                    car3.num, car3.gas);
21:
22:         printf("(car3) num > ");
23:         scanf("%d", &car3.num);
24:         printf("(car3) gas > ");
25:         scanf("%lf", &car3.gas);
26:         printf("(car3) num: %d, gas: %f\n",
                      car3.num, car3.gas);
27:
```

メンバのアドレスを指定する場合はアドレス演算子もあわせて使う
（メンバnumはint型なので%dを指定し、アドレス演算子を使う）

構造体変数同士を「=」で代入すると
全てのメンバが一度にコピーされる (p.360)

```
28:         car2 = car1;
29:         printf("(car2) num: %d, gas: %f\n”,
                    car2.num, car2.gas);
30:
31:         return 0;
32:   }
```

構造体の場合、変数同士の代入は一回の代入処理で、全てのメンバの値が一度にコピーされます。この例では、car1をcar2へ代入することで、次のようにメンバの値が変わります。



typedefで名前を付ける（p.356）

構造体型を宣言する際に、typedefというキーワードを使うことで、構造体型の名前を「struct Car」の代わりに「Car」という名前に短縮できるようになります。

【例1】プログラムの3〜6行目を以下のように変更する

```
typedef struct Car {
    int num;
    double gas;
} Car;
```

typedefで名前を付ける（p.356）

構造体変数を宣言する際にも、「struct Car」の代わりに「Car」という型名を使用できるようになります。

【例2】プログラムの10〜11行目を以下のように変更する

```
Car car1, car2;
Car car3 = {1357, 20.3};
```

ポインタでの参照

構造体変数を参照するポインタを宣言する場合は、ポインタの型名を構造体型にします。（int型やchar型に対するポインタと同様）

【例3】プログラムの11行目以降に以下の宣言を追加する

```
/* typedefを使っていれば、Car *ptr1; でも可能 */

struct Car *ptr1;
```

ポインタptr1が構造体変数car2を参照する

【例4】プログラムの29行目以降に以下の処理を追加する

```
ptr1 = &car2;

printf("(ptr1) num: %d, gas: %f\n",
        (*ptr1).num, (*ptr1).gas);

printf("(ptr1) num: %d, gas: %f\n”,
        ptr1->num, ptr1->gas);
```

まず「*ptr1」部分の**間接参照演算子**を演算して、次に
「.gas」部分の**ドット演算子**を演算する
従って「ptr1が参照先のメンバgasの値」を意味している

間接参照演算子とドット演算子を組み合わせて毎回書くのは
大変なので、代わりに「->」という**アロー演算子**が使える
(p.371)

引数として使う（p.368）

関数の引数に構造体を使う場合、これまでの変数と同様に値渡しと参照渡しが可能です。

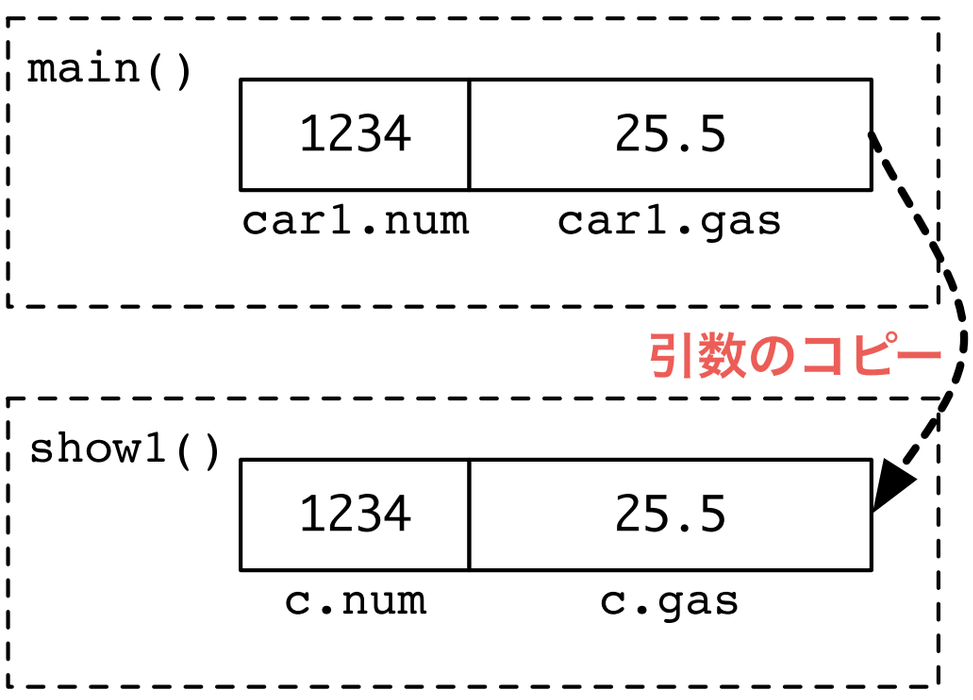
【例5】プログラムに以下の関数を追加する

```
/* 値渡し（仮引数cに実引数の構造体がコピーされる）*/
void show1(Car c, char *s)
{
    printf("(%s) num: %d, gas: %f\n",
            s, c.num, c.gas);
}

/* 参照渡し（仮引数pCが実引数の構造体を参照する）*/
void show2(Car *pC, char *s)
{
    printf("(%s) num: %d, gas: %f\n",
            s, pC->num, pC->gas);
}
```

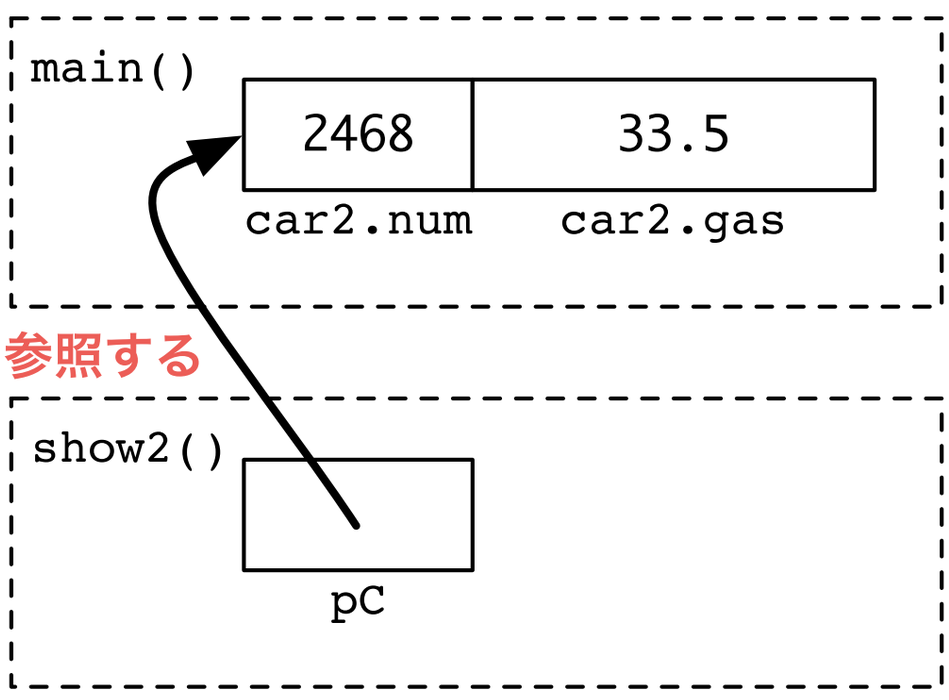
【例6】プログラムのmain()の最後に以下の関数を追加する

```
/* 実引数car1の内容が、仮引数へコピーされる */
show1(car1, "car1");

/* 実引数car2のアドレスが、仮引数へコピーされる */
show2(&car2, "car2");
```

関数呼出し時の仮引数と実引数の様子



【値渡し】実引数car1の値が仮引数cへコピー



【参照渡し】実引数car1を仮引数pCが参照

構造体は参照渡しをするのが一般的

【準備】

最初のスライドで示したサンプルプログラムをコピーして動作を確認してみましょう。

なお、サンプルプログラムは、以下のようにcpコマンドを使って、自分の所にコピーして使うことができます。

```
$  cp  /usr/local/common/kogai/prog2d/kiso210-1.c  .（←ここにピリオド）
```

【練習10-1】

サンプルプログラムを、例1, 例2で示したように、typedefを使ったプログラムに変更して実行結果を確認してみましょう。

（実行結果はサンプルプログラムと同じです。）

【練習10-2】

練習10-1のプログラムを、例3, 例4で示したように、ポインタを使ったプログラムに変更して実行結果を確認してみましょう。

【練習10-3】

練習10-2のプログラムを、例5, 例6で示したように、関数を使った処理をプログラムに追加して実行結果を確認してみましょう。

【課題10-1】

「引数が参照しているCarの構造体を初期状態にする」関数clear()を作成してください。

[この関数のプロトタイプ宣言]

```
void clear(Car *pC);

  /* 仮引数のポインタpCが参照している構造体のメンバを
     以下のように変更する */
  /*    ● メンバnumを 4567 にする */
  /*    ● メンバgasを 40 にする */
```

【課題10-1】

[mainでの処理]

```
Car car1;
car1.num = 1234;
car1.gas = 25.5;
show2(&car1, "current car1");
clear(&car1);
show2(&car1, "cleared car1”);
```

[実行結果]

```
(current car1) num: 1234, gas: 25.500000
(cleared car1) num: 4567, gas: 40.000000
```

【課題10-2】

「引数が参照している2つのCarの構造体の中身を交換する」関数swap()を作成してください。

[この関数のプロトタイプ宣言]

```
void swap(Car *p1, Car *p2);

  /* 交換用に使うCarの構造体変数を用意する */
  /* 仮引数のポインタが参照している構造体の代入をして入れ替える
       （後期 第3週 例3の型をint型からCar型へと変更する） */
  /* （またはnumとgasをメンバ毎に代入して入れ替えることもできる） */
```

【課題10-2】

[mainでの処理]

```
 Car car1 = {4567, 40.0};
 Car car2 = {5678, 10.5};
 show2(&car1, "current car1");
 show2(&car2, "current car2");
 swap(&car1, &car2);
 show2(&car1, "swapped car1");
 show2(&car2, "swapped car2”);
```

[実行結果]

```
 (current car1) num: 4567, gas: 40.000000
 (current car2) num: 5678, gas: 10.500000
 (swapped car1) num: 5678, gas: 10.500000
 (swapped car2) num: 4567, gas: 40.000000
```

【課題10-3】

「引数が参照している2つのCarの構造体のメンバgasを比較する」関数compare_gas()を作成してください。

[この関数のプロトタイプ宣言]

```
int compare_gas(Car *p1, Car *p2);

  /* ポインタp1とp2が参照している構造体のメンバgasを比較し、
        その結果によって戻り値を以下のように決める */
  /* ● p1のgasの方が大きかったら、戻り値を 1 とする */
  /* ● p2のgasの方が大きかったら、戻り値を -1 とする */
  /* ● p1とp2のgasが等しかったら、戻り値を 0 とする */
```

【課題10-3】

[mainでの処理]

```
 Car car1 = {4567, 30.5};
 Car car2 = {5678, 20.0};
 printf("compare: %d\n", compare_gas(&car1, &car2));
 car2.gas = 40.5;
 printf("compare: %d\n", compare_gas(&car1, &car2));
 car2.gas = 30.5;
 printf("compare: %d\n", compare_gas(&car1, &car2));
```

[実行結果]

```
 compare: 1
 compare: -1
 compare: 0
```

【課題10-4】

次のような時間（時と分）の情報を持つ構造体型を宣言します。

```
typedef struct Time {
     int hour;
     int minute;
} Time;
```

この構造体の情報を画面に出力する関数show()を作成してください。

[この関数のプロトタイプ宣言]

```
void show(Time *t, char *s);

   /* Carの構造体の出力の処理を参考に作成できる */
   /* 出力の書式は実行結果の様子を参照 */
```

【課題10-4】

[mainでの処理]

```
 Time t1 = {40, 50};   /* 40時間50分 */
 Time t2 = {30, 40};   /* 30時間40分 */
 show(&t1, "initial t1”);
```

[実行結果]

```
 (initial t1) 40:50
```

まだ余裕のある人は… 【課題10-5】

「引数が参照している2つのTimeの構造体が表す時間を加算する」関数add_time()を作成してください。

[この関数のプロトタイプ宣言]

```
void add_time(Time *t1, Time *t2);

  /* メンバhour同士とminute同士を加算する */
  /* メンバhour同士とminute同士を加算する */
  /* minuteで60分を超える場合は、
     hourへ繰り上げる（minuteと60との商をhourへ加える） */
```

【課題10-5】

```
[mainでの処理(課題10-4の続き)]
 add_time(&t1, &t2);     /* 40:50 と 30:40 を加算する */
 show(&t1, "added t1”);

[実行結果]
 (added t1) 71:30
```

まだまだ余裕のある人は…【課題10-6】

「引数が参照している2つのTimeの構造体が表す時間を比較する」関数compare_time()を作成してください。

[この関数のプロトタイプ宣言]

```
int compare_time(Time *t1, Time *t2);
  /* t1とt2が参照している構造体の時間を比較し、
        その結果によって戻り値を以下のように決める */
  /* ● t1の時間の方が大きかったら、戻り値を 1 とする */
  /* ● t2の時間の方が大きかったら、戻り値を -1 とする */
  /* ● t1とt2の時間が等しかったら、戻り値を 0 とする */
  /* 時間の比較は、hour同士を比較する処理と、
        minute同士を比較する処理をifを組み合わせて作れる
        （または、まず、分に値を変換してから比較することでも作れる） */
```

【課題10-6】

[mainでの処理]

```
Time t1, t2;
t1.hour = 20;  t1.minute = 10;
t2.hour = 15;  t2.minute = 50;
printf("compare: %d\n", compare_time(&t1, &t2));
t2.hour = 20; t2.minute = 55;
printf("compare: %d\n", compare_time(&t1, &t2));
t2.hour = 20; t2.minute = 10;
printf("compare: %d\n", compare_time(&t1, &t2));
```

[実行結果]

```
compare: 1
compare: -1
compare: 0
```

小テストについて

前回の小テストを実施予定

（時間があれば・・・）