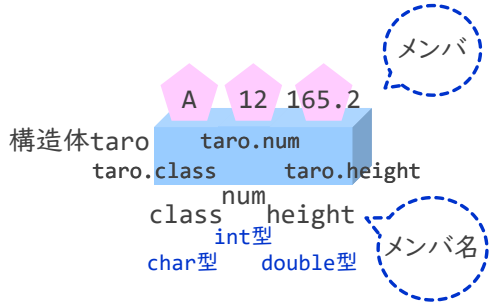
プログラミング基礎演習A

構造体

構造体

- 異なる型のデータを入れる複数の「箱」の集合
- 構造体のメンバ ： 各箱に入る値
  他の変数同様に代入や演算に使用可
  構造体名.メンバ名と表記

関数の引数や戻り値にも使用可



構造体の定義

```
struct タグ名
{
    型名 メンバ名;
    型名 メンバ名;
    …
};
```

- 構造体が持つデータを定義する

```
struct student
{
    char class;
    int num;
    double height;
};
```


構造体変数の宣言

```
struct タグ名 構造体名
```

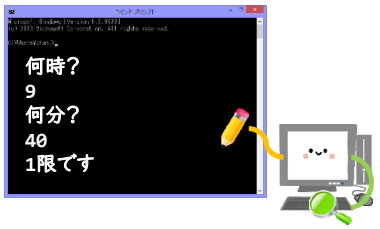
↳ int,doubleなどの型名と同様に使用

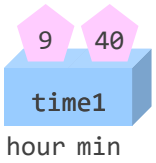
- 使用する構造体変数の領域を確保する

```
int main()
{
    struct student taro;
    struct student group_a[3];

    ...
}
```

プログラム例(1)

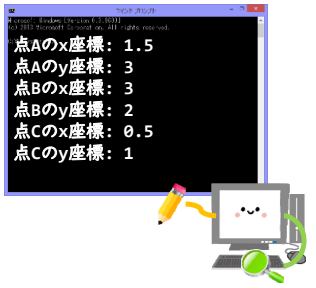
```c
#include <stdio.h>

struct tag_time
{
    int hour;
    int min;
};

int main()
{
    struct tag_time time1;

    printf("何時?¥n");
    scanf("%d", &(time1.hour));
    printf("何分?¥n");
    scanf("%d", &(time1.min));
    if((time1.hour == 9) ||
       ((time1.hour == 10) && (time1.min <= 40))) {
        printf("1限です¥n");
    } else {
        printf("1限以外の時間です¥n");
    }
}
```

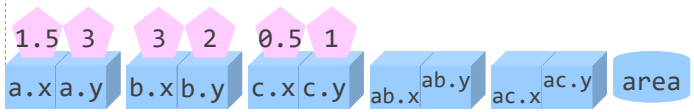




プログラム例(2)

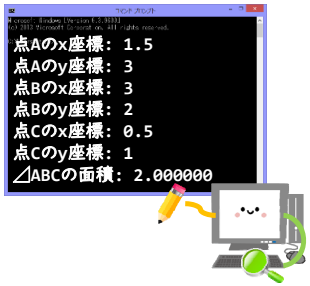
```c
#include <stdio.h>

struct tag_vector2
{
    double x;
    double y;
};

int main()
{
    struct tag_vector2 a;
    struct tag_vector2 b;
    struct tag_vector2 c;
    struct tag_vector2 ab;
    struct tag_vector2 ac;
    double area;

    printf("点Aのx座標: ");
    scanf("%lf", &(a.x));
    printf("点Aのy座標: ");
    scanf("%lf", &(a.y));
    printf("点Bのx座標: ");
    scanf("%lf", &(b.x));
    printf("点Bのy座標: ");
    scanf("%lf", &(b.y));
    printf("点Cのx座標: ");
    scanf("%lf", &(c.x));
    printf("点Cのy座標: ");
    scanf("%lf", &(c.y));
```

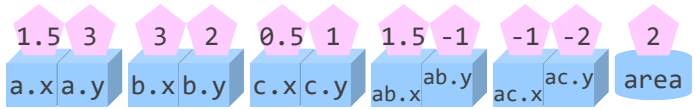




プログラム例(2)

```c
    ab.x = b.x - a.x;
    ab.y = b.y - a.y;
    ac.x = c.x - a.x;
    ac.y = c.y - a.y;
    area = 0.5 * (ab.x * ac.y - ab.y * ac.x);
    if(area < 0) {
        area = -area;
    }
    printf("◿ABCの面積: %f¥n", area);
}
```





構造体の初期化

struct タグ名 構造体名 = {メンバ, メンバ, …}

- 変数の宣言時のみ初期化可能

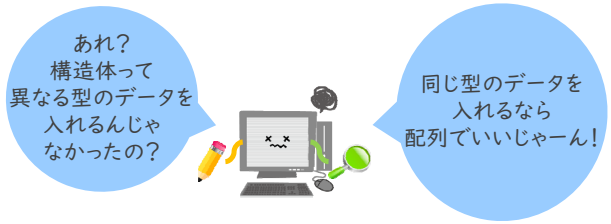
```c
struct tag_time a = {9, 20};
```

➡ a.hour = 9, a.min = 20

```c
struct tag_time b[3] = {{9, 0}, {10, 50}, {13, 20}};
```

➡
- b[0].hour = 9, b[0].min = 0
- b[1].hour = 10, b[1].min = 50
- b[2].hour = 13, b[2].min = 20

ぱそ子のつぶやき…



そんなことはありません！

構造体は配列と違って代入できるんです！

```
struct tag_time a = {9, 20};
struct tag_time b;
b = a;
```

関数での活用も簡単！

プログラム例(3)

```
#include <stdio.h>

struct tag_time
{
    int hour;
    int min;
};

int total_min(struct tag_time d)
{
    int m;

    m = d.hour * 60 + d.min;
    return m;
}
```

関数の引数に使用

```
struct tag_time oclock(int h)
{
    struct tag_time r;

    r.hour = h;
    r.min = 0;
    return r;
}
```

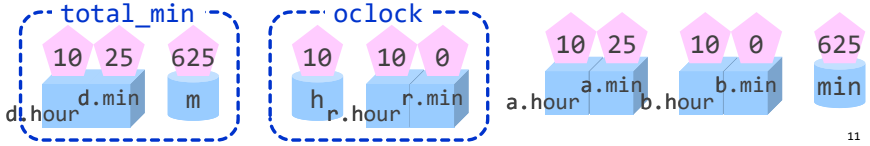
関数の戻り値に使用

プログラム例(3)

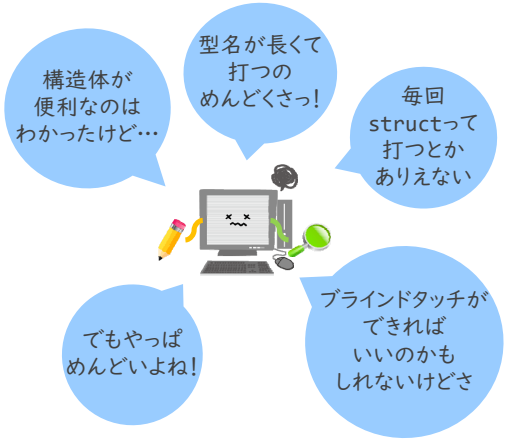
```
int main()
{
    struct tag_time a;
    struct tag_time b;
    int min;

    a.hour = 10;
    a.min = 25;
    min = total_min(a);
    printf("計%d分¥n", min);
    b = oclock(10);
    printf("%d時¥n", b.hour);
    printf("%d分¥n", b.min);
}
```

引数として渡す

戻り値として受け取る



ぱそ子のつぶやき…



typedefを使った略記

```
typedef struct
{
    型名 メンバ名;
    型名 メンバ名;
    …
} 型名;
```

```
typedef struct
{
    int hour;
    int min;
} my_time;
```

と定義すると `my_time a;` と宣言できる!

プログラム例(4)

```
#include <stdio.h>

typedef struct
{
    int hour;
    int min;
} my_time;

int total_min(my_time d)
{
    int m;

    m = d.hour * 60 + d.min;
    return m;
}
```

```
my_time oclock(int h)
{
    my_time r;

    r.hour = h;
    r.min = 0;
    return r;
}
```

```
int main()
{
    my_time a;
    my_time b;
    int min;

    a.hour = 10;
    a.min = 25;
    min = total_min(a);
    printf("計%d分¥n", min);
    b = oclock(10);
    printf("%d時¥n", b.hour);
    printf("%d分¥n", b.min);
}
```

演習

xy平面上の2つの点A, Bのx座標とy座標を入力すると, 2点間の距離が表示されるプログラムを作成せよ. ただし, 点のx座標とy座標は構造体pointに格納すること. 構造体pointはtypedef宣言をして, structを書かずに使えるようにすること. また, 2点間の距離は次に示す関数calcDisで求めること. 関数はプロトタイプ宣言をして, メイン関数の後に定義を書くこと.

関数calcDis
　引数　：1つめの点の座標 <構造体point>
　　2つめの点の座標 <構造体point>
　戻り値 ：2点間の距離 <double型>

プログラム名はe11とすること.

<実行例>

```
点Aのx座標:5
点Aのy座標:8
点Bのx座標:6
点Bのy座標:2
点Aと点Bの距離は
6.082763です
```

赤字は実行時にキーボードから入力する部分

演習　～手順～

1. 中身が空のメイン関数, 必要なヘッダファイルを書く
2. 定義する構造体の仕様は?
   - → typedef名はpoint
   - → メンバは2つの実数
   - → typedefを使って定義する
3. 作る関数の仕様は?
   - → 関数名はcalc_distance
   - → 2つの引数は構造体point
   - → 戻り値は実数
   - → プロトタイプ宣言を書く
   - → 中身が空の関数を書く

```
#include <stdio.h>
#include <math.h>

typedef struct
{
    double x;
    double y;
} point;

double calcDis(point point1, point point2);

int main()
{
}

double calcDis(point point1, point point2)
{
}
```

演習 ～手順～

4. メイン関数で必要な変数は？
   → 点Aと点Bの座標を代入する変数
   → 2点間の距離を代入する変数
5. メイン関数の処理の順番は？
   → 点Aの座標を読み込む（printf & scanf）
   → 点Bの座標を読み込む（printf & scanf）
   → 2点間の距離を計算する（関数呼出し）
   → 2点間の距離を表示する（printf）
6. calc_distance関数で必要な変数は？
   → 2点間の距離を代入する変数
7. calc_distance関数の処理の順番は？
   → 2点間の距離を求める