構造体

複数の変数の扱い方

例) 二つの点の中点を求める

- 一次元(直線状の点)なら

```
median (float x1, float x2, float *cx);
```

- 二次元(平面上の点)なら

```
median (float x1, float y1, float x2, float y2, float *cx, float *cy);
```

- 三次元(空間中の点)なら…

```
median (float x1, float y1, float z1, float x2, float y2, float z2
                float *cx, float *cy, float *cz);
```

解決策：配列を用いる　→　例）p[2], p[3]
しかし，変数の型が一致しない場合は？
例）int x, float y, double z;

構造体

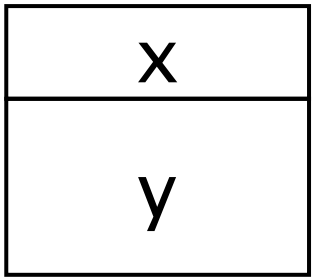
- 構造体
  - 複数のデータからなるデータ
    - それぞれのデータをメンバ変数という

例) 点は複数の座標から成るのでまとめて扱いたい

```
struct Point {
        int x;
        float y;
};
```

最後の;に注意

```
struct Point a;
```



a

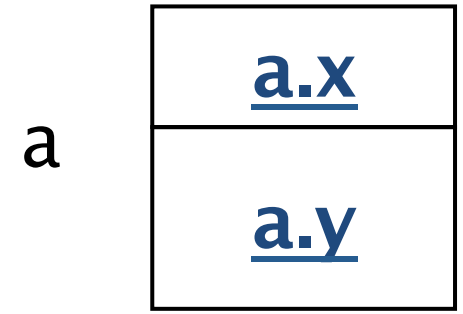
二つのデータのかたまりにaという名前が付く

構造体メンバ変数の参照

- 変数の中に複数のデータがある
  - どうやって、それぞれをアクセスするか？
  - 構造体変数名. メンバ変数名

```
struct Point {
        int x;
        float y;
};
```

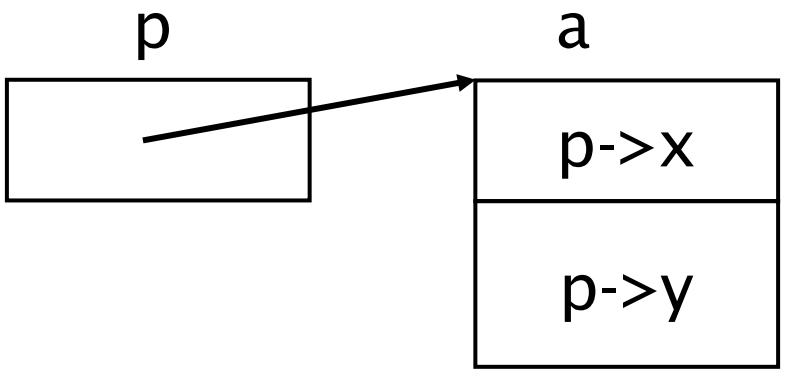
```
struct Point a;
```



構造体メンバの参照

- 構造体へのポインタがある場合
  - 構造体ポインタ名ー＞メンバ変数名
    - ポインタ参照(*)がないことに注意
    - *p->x ではなく p->x でメンバ変数参照

```
struct Point {
        int x;
        float y;
};
struct Point a;
struct Point *p = &a;
```



*p->x と(*p).xは同じものを参照*

構造体を使う利点

- 意味的なかたまりを扱えるので、関数に渡す変数の数を，見かけ上少なくできる
- 似たデータ構造の拡張が構造体の宣言とその構造体をあつかう関数の変更だけで実現できる
  - 機能の局所化

例) 二次元→三次元

- 構造体のメンバ変数を増やす
- 計算する関数の処理を増やす

```
median(struct Point p1, struct Point p2
          struct Point cp) {
          …;
}
```

```
struct Point {
          int x;
          float y;
};
```

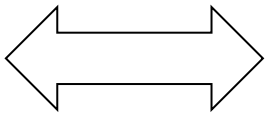
↓

```
struct Point {
          int x;
          float y;
          double z;
};
```

構造体の各種使用法

```
struct Point {
        int x;
        float y;
};
int main () {
        struct Point p0, p1;
```

同じ使用法



```
int main () {
        struct Point {
                int x;
                float y;
        } p0, p1;
```

```
struct Point {
        int x;
        float y;
};
int main () {
struct Point *p0, *p1;
p0 = (struct Point *) malloc (sizeof (struct Point) );
```

動的な領域確保

中点計算の実装例

```
void median (struct Point p0, struct Point p1,
                                        struct Point *cp) {
        cp->x = (p0.x + p1.x) / 2;
        cp->y = 0.5 * (p0.y + p1.y);
        cp->z = 0.5 * (p0.z + p1.z);
}
struct Point * median (struct Point p0, struct Point p1) {
        struct Point *cp = (struct Point *) malloc (
                                        sizeof (struct Point));
        cp->x = (p0.x + p1.x) / 2;
        cp->y = 0.5 * (p0.y + p1.y);
        cp->z = 0.5 * (p0.z + p1.z);
        return cp;
}
```

構造体の使用例

- 健康管理データに構造体を使う
  - 身長，体重の値を記録
  - 身長・体重の値から肥満度(BMI)を計算する
    - 同じ構造体に変数（メンバ変数）を確保する

```
struct Health {
        char  *name; /* 対象者の氏名 */
        float  height, weight; /* 身長・体重 */
        float  BMI; /* 計算した肥満度 */
};
```

構造体を引数に取る関数

```
/* BMI指数＝体重(kg)÷{身長(m)Ｘ 身長(m)} */
/* 構造体のコピーを渡す場合 */
float calcBMI (struct Health data) {
    return data.weight / (data.height*data.height);
}
/* 構造体のアドレスをポインタに渡す場合 */
void calcBMI_P (struct Health *data) {
   data->BMI = data->weight / (data->height*data->height);
}
```

構造体の使用例

```
struct Health data; /* 構造体データ */

data.name = “Toyohashi Taro”;
data.height = 1.7;
data.weight = 55;

data.BMI = calcBMI (data);/* 肥満度を返す場合 */
または,
calcBMI_P (&data);/* 関数内で肥満度を更新する場合 */
```

構造体の配列

- 同じ構造体データを複数処理する場合は...

```
struct Health data[100];
for (i = 0; i < 100; i++) {
        data[i].BMI = calcBMI (data[i]);
}
```

- 構造体の配列を関数に渡す場合

```
void calcAllBMI (struct Health data[]) {
  for (i = 0; i < 100; i++) {
        data[i].BMI = calcBMI (data[i]);
        /* calcBMI_P (data + i); */
  }
```

構造体の配列

- 動的な領域確保は...

```
struct Health *dataPtr; /* 構造体へのポインタ */
/* 100人分のデータを確保 */
dataPtr = (struct Health *)
                malloc (sizeof (struct Health) * 100);
```

- 以後，配列と同様に扱える

```
for (i = 0; i < 100; i++) {
    dataPtr[i].BMI = calcBMI (dataPtr[i]);
    /* *(dataPtr + i).BMI, (dataPtr + i)->BMI */
}
```

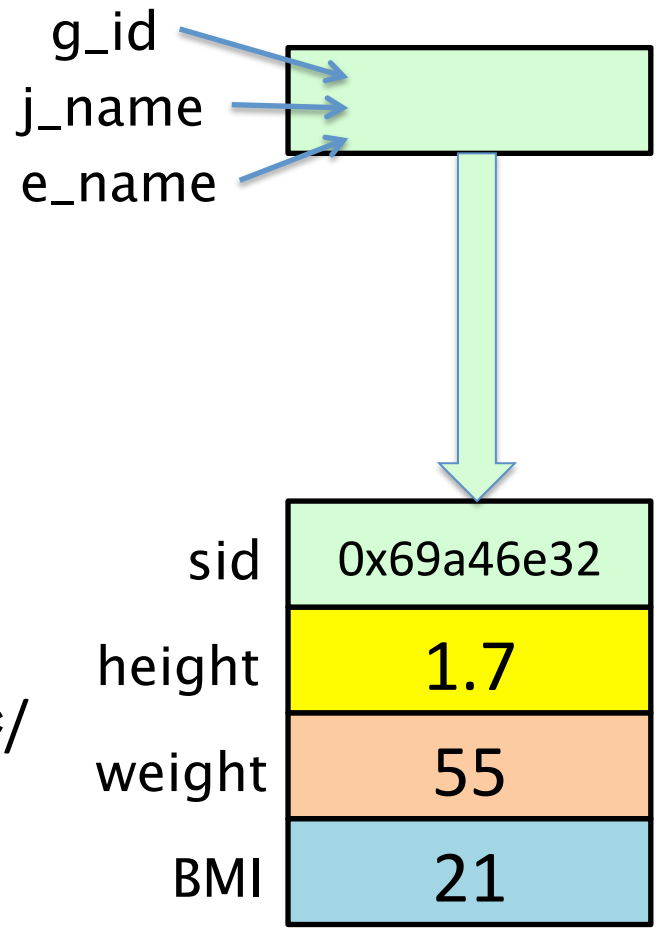
共用体

- 構造体の特殊な形態
- メンバ変数が同じメモリ領域にオーバラップして確保される
  - メンバ変数のどれか一つを選択的に使用する
  - メモリ領域を節約する

共用体の宣言例

```
union StudentID { // 学生のIDデータ
    char *g_id; // 学籍番号
    char *j_name; // 日本語表記
    char *e_name; // ローマ字表記
};

struct Health {
    union StudentID sid; /* IDデータ */
    float  height, weight; /* 身長・体重 */
    float  BMI; /* 計算した肥満度 */
};
```



共用体の使用例

```
int main () {
    struct Health health;

    health.sid.g_id = "123456";
    health.sid.j_name = “豊橋太郎”; // g_id の値に上書き
    health.sid.e_name = “Taro Toyohashi”; // j_name に上書き
    health.height = 170.0;
  ...
```