#10
構造体
2022 年度 / プログラミング及び実習 III

角川裕次

龍谷大学　先端理工学部

もくじ

今回 (#10) の内容： シラバスでの該当部分

小テーマ: **構造体**

第 16 回： **構造体**
第 17 回： **構造体へのポインタ型**

重要概念リスト

- **構造体**：**関連情報をひとまとめにする新たな型**
- **派生型**
  - **配列型** Type a[n];
  - **構造体型** `struct { Type1 m1; Type2 m2; ... }` a;
  - **共用体型** `union { Type1 m1; Type2 m2; ... }` a;
  - **関数型** Type a(Type1 m1, Type2 m2, ...){ ... }
  - **ポインタ型** Type *a;
- **. 演算子により構造体のメンバをアクセス**
  - **例**：`point.x`
- **-> 演算子によりポインタで指される構造体へのメンバをアクセス**
  - **例**：`p->x`
- **構造体は代入が可能**
- **配列は代入が不可能**

今回の実習・課題 (manaba へ提出)

実習内容と課題内容は講義途中に提示します

(**作成したファイル類は** manaba **に提出**)

第 12-1 節　構造体

★重要★

□ データの関連性 p.330

List 12-1 は学生情報 (名前と身長) を身長に関して昇順にソート
学生情報 (名前と身長) は以下の形で記述 (プログラムコード詳細は省略)

```
int  height[] =           {178,    175,      173,
                           165,    179};
char name[][NAME_LEN] = {"Sato", "Sanaka", "Takao",
                         "Mike", "Masaki"};
```

問題点: 学生に関する情報 (名前と身長) の記述が分離してわかりにくい

- 関連情報が別々に記述されているので修正・改造が困難 (保守性低)
- ソースコード上から関連性を読み取るのが困難 (可読性低)

解決法: 構造体 (structure) の導入:
ソースコード上でひとまとめにする記述をして保守性・可読性が向上

- **保守性向上**: 関連する情報をひとまとめに記述
- **可読性向上**: ソースコード上から関連性を容易に読み取れる

★重要★

□ 構造体 p.332

例：学生を表す情報の項目

- **名前，身長，体重**

構造体によるデータ型の定義

```
struct student {
  char    name[64];  /*名前*/
  int     height;    /*身長*/
  float   weight;    /*体重*/
};
```

(**新たなデータ型** `struct student` **を定義するだけ；変数宣言はまだ**)

用語

- `student`：**構造体タグ** (**構造体のおなまえ**)
- `name`, etc.：**メンバ** (**構造体中の要素**)

構造体の変数宣言

★重要★

変数 sanaka を宣言

```
struct student sanaka;
```

- **構造体 struct student は定義済みであること**

複数の変数の宣言

```
struct student s1, s2, s3, s4;
struct student s5, s6;
```

- **変数を 6 つ宣言**

構造体のタグ名の定義をしない変数宣言も可能

```
struct {
  int    x;
  long   y;
  double z;
} a, b;
```

- **不便：タグ名が定義されないので他の場所では使えない**

★重要★

□ 構造体のメンバと . 演算子 p.334

構造体のメンバへのアクセス: ドット演算子 . を使用

例 : `sanaka.height = 175;`

List 12-2 (**部分**) : **学生情報を表す構造体の使用例**

```
int main(void)
{
  struct student sanaka;
  strcpy(sanaka.name, "Sanaka"); /* 名前 */
  sanaka.height = 175;           /* 身長 */
  sanaka.weight = 62.5;          /* 体重 */
  printf("氏 名 = %s\n",    sanaka.name);
  printf("身 長 = %d\n",    sanaka.height);
  printf("体 重 = %.1f\n", sanaka.weight);
  return 0;
}
```

□ メンバの初期化 p.335

宣言時に初期値を指定することで構造体変数の初期化が可能

List 12-3 (**部分**) : **学生情報を表す構造体の使用例**

```
int main(void)
{
  struct student takao = {"Takao", 173};
  printf("氏名 = %s\n",    takao.name);
  printf("身長 = %d\n",    takao.height);
  printf("体重 = %.1f\n", takao.weight);
  return 0;
}
```

初期値はメンバの順に書き連ねて { と } で囲む

```
{"Takao", 173}
```

初期化子の記述がなければ 0 **に初期化される**

- takao.weight **の値は** 0 **に初期化**

□ 構造体のメンバと -> 演算子 p.336

間接演算子とドット演算子を使用した例

List 12-4 (**部分**) : **構造体へのポインタとメンバへのアクセス**

```
/* sが指す学生の体重が0以下であれば標準体重を代入 */
void set_stdweight(struct student *s)
{
  if ((*s).weight <= 0)
      (*s).weight = ((*s).height - 100) * 0.9;
}
int main(void)
{
  struct student takao = {"Takao", 173};
  set_stdweight(&takao);
  printf("氏名 = %s\n",   takao.name);
  printf("身長 = %d\n",   takao.height);
  printf("体重 = %.1f\n", takao.weight);
  return 0;
}
```

メンバへのアクセス：間接演算子とドット演算子

アドレス演算子 `&` の使用例：構造体変数 `takao` へのポインタを得る

```
&takao
```

関数呼び出し例：ポインタを引数にして呼び出す

```
set_stdweight(&takao);
```

間接演算子とドット演算子の使用例：
ポインタ変数 `s` が指す構造体のメンバ `height`

```
(*s).height
```

- **間接演算子 `*` でポインタから構造体を得る**
- **ドット演算子 `.` で構造体のメンバの値を参照**

★重要★

メンバへのアクセス：-> 演算子

メンバアクセス演算子 ->で構造体のメンバを表す

例：ポインタ変数 std が指す構造体のメンバ height

```
s->height
```

- (*s).height と同等

List 12-4 の書き換え例 (部分)

```
/* s が指す学生の体重が 0 以下であれば標準体重を代入 */
void set_stdweight(struct student *s)
{
  if (s->weight <= 0)
      s->weight = (s->height - 100) * 0.9;
}
```

□ 構造体と typedef　p.338

構造体の型名を typedef により別名を付けることができる

List 12-5 (**部分**)

```
typedef struct student {
  char   name[NAME_LEN]; /* 名前 */
  int    height;         /* 身長 */
  float  weight;         /* 体重 */
} Student;
```

型 Student : 構造体 struct student 型の別名として定義

```
typedef struct student {
  .... 略 ...
} Student;
```

構造体への typedef の例

List 12-5 (**部分**)

```
typedef struct student {
  char  name[NAME_LEN]; /* 名前 */
  int   height;         /* 身長 */
  float weight;         /* 体重 */
} Student;

void set_stdweight(Student *s)
{
  if (s->weight <= 0)
      s->weight = (s->height - 100) * 0.9;
}
...
```

□ 構造体とプログラム p.339

対象の属性をひとまとめ：関連性がひと目で分かる

- **ソースコードの可読性・保守性を向上できる**
- **ソースコードを書く時・読む時わかりやすく間違いしにくい**
- **あとから属性を追加するときわかりやすい**

```
struct student {
  char     name[NAMELEN];
  double   height;
  double   weight;
} student[N];
```

対象の属性がばらばらに記述されると関連性が分からない

- **ソースコードの可読性・保守性が悪い**

```
char name[N][NAMELEN];
double  height[N];
double  weight[N];
```

□ 集成体型 (aggregate type) p.340

いくつかのデータ型をひとまとめにしたデータ型

- **構造体 と 配列 はともに集成体型**

配列と構造体の違い 1：要素型

- **配列　：同じ型のデータ型の集成**
- **構造体：同じ型とは限らないデータ型の集成**

配列と構造体の違い 2：代入

- **配列　：不可能**
- **構造体：全メンバを一度に代入が可能**

□ 構造体の値を返却する関数 p.340

関数の返却値型として構造体を指定可能

- **構造体は代入が可能なので**
- **(配列はだめ; 代入が不可能なので)**

List 12-6 (**部分**) : **構造体を返す関数の例**

```
struct xyz {
  int    x;
  long   y;
  double z;
};
struct xyz xyz_of(int x, long y, double z)
{
  struct xyz temp;
  temp.x = x;
  temp.y = y;
  temp.z = z;
  return temp;
}
```

- **局所変数** `temp` **に値を代入**
- **その構造体を値として関数値として返す**

構造体の値を返却する関数の呼び出し例

List 12-6 (**部分**)

```
int main(void)
{
  struct xyz s;
  s = xyz_of(12, 7654321, 35.689);
  printf("xyz.x = %d\n",  s.x);
  printf("xyz.y = %ld\n", s.y);
  printf("xyz.z = %f\n",  s.z);
  return 0;
}
```

実行結果

```
xyz.x = 12
xyz.y = 7654321
xyz.z = 35.689000
```

□ 名前空間 p.341

名前空間の分類 (4 種類)

1. **ラベル名 (goto 文でのジャンプ先)**
2. **タグ名 (構造体の名前)**
3. **メンバ名 (構造体の構成要素)**
4. **一般的な識別子 (変数名/型名)**

名前空間が異なれば同じ名前を使って良い (混乱しない)

```
int main(void) {
  struct x {   /*タグ名*/
    int x;     /*メンバ名*/
    int y;     /*メンバ名*/
  } x;         /*識別子*/
x:             /*ラベル名*/
  x.x = 1;     /*変数名.メンバ名*/
  x.y = 5;     /*変数名.メンバ名*/
  return 0;
}
```

□ 構造体の配列 p.342

構造体の配列を宣言できる

- **他の型の配列と同様**

List 12-7 (**部分**) : **構造体の定義と配列の宣言**

```
typedef struct {
  char  name[NAME_LEN]; /* 名前 */
  int   height;         /* 身長 */
  float weight;         /* 体重 */
} Student;
int main(void)
{
  Student std[] = {  // ★構造体の配列★
    { "Sato",   178, 61.2},  /* 佐藤君 */
    { "Sanaka", 175, 62.5},  /* 佐中君 */
    { "Takao",  173, 86.2},  /* 高尾君 */
    { "Mike",   165, 72.3},  /* Mike君 */
    { "Masaki", 179, 77.5},  /* 真崎君 */
  };
  ... 略 ...
}
```

構造体の配列の宣言の例

初期化子を伴う宣言 (要素数は初期化子より自動決定)

```
Student std[] = {  // ★構造体の配列★
  { "Sato",   178, 61.2},  /* 佐藤君 */
  { "Sanaka", 175, 62.5},  /* 佐中君 */
  { "Takao",  173, 86.2},  /* 高尾君 */
  { "Mike",   165, 72.3},  /* Mike君 */
  { "Masaki", 179, 77.5},  /* 真崎君 */
};
```

要素数を指定した宣言 (初期値は不指定)

```
Student std[10];
```

構造体の値の交換

List 12-7 (**部分**) : 2 **つ構造体の値を交換**

```
void swap_Student(Student *x, Student *y)
{
  Student temp = *x;
  *x = *y;
  *y = temp;
}
```

構造体から構造体への代入ができる

(**注意**: **配列から配列への代入はできないのでしたよね**...)

構造体の配列のソート

List 12-7 (**部分**) : 5 **人のデータを身長でソート**

```
void sort_by_height(Student a[], int n)
{
  for (int i = 0; i < n - 1; i++) {
    for (int j = n - 1; j > i; j--)
      if (a[j - 1].height > a[j].height)
        swap_Student(&a[j - 1], &a[j]);
  }
}
```

バブルソート

- **身長の大小を比較**
- **構造体の内容をまるごと交換**

★重要★

□ 派生型 p.342

派生型 (derived type)

既存のデータ型から作り出された新たなデータ型

- 無数に新たな定義が可能

派生により作られる型

- 配列型：要素の型と要素の数から決まる
- 構造体型：列挙されたメンバのリストから決まる
- 共用体型：重なり合うメンバのリストから決まる
- 関数型：返却値型と仮引数の型のリストから決まる
- ポインタ型：オブジェクトまたは関数の型から決まる

第 12-2 節　メンバとしての構造体

□ 座標を表す構造体 p.344

List 12-8 (抜粋) より

x と y の 2 つの値で 2 次元空間の点を表す構造体

```
typedef struct  {
  double x;       /*  X 座標  */
  double y;       /*  Y 座標  */
} Point;
```

- タグ名は与えず typedef により別名の型を定義

2 点間の距離を求める関数

```
#define sqr(n)  ((n) * (n))    /*  2 乗値を求める  */
double distance_of(Point p1, Point p2)
{
  return sqrt(sqr(p1.x - p2.x) + sqr(p1.y - p2.y));
}
```

座標を表す構造体の利用例

List 12-8 (抜粋) より：main 関数

```
int main(void)
{
  Point crnt, dest;
  printf("現在地のX座標：");  scanf("%lf", &crnt.x);
  printf("       Y座標：");  scanf("%lf", &crnt.y);
  printf("目的値のX座標：");  scanf("%lf", &dest.x);
  printf("       Y座標：");  scanf("%lf", &dest.y);
  printf("目的値までの距離は%.2fです。\n",
         distance_of(crnt, dest));
  return 0;
}
```

□ 構造体のメンバを持つ構造体 p.345

Fig 12-13 **メンバに構造体を持つ構造体**

```
typedef struct  {
  double x;      /*  X 座標  */
  double y;      /*  Y 座標  */
} Point;

typedef struct {
  Point pt;      /*現在位置*/
  double fuel;   /*残り燃料*/
} Car;
```

変数宣言の例 (Car **構造体の中に** Point **構造体が含まれている**)

```
Car c;
```

メンバへのアクセス例

```
c.pt.x = 10.0;
c.pt.y = 20.0;
c.fuel = 100.0;
```

List 12-9 概説 (1)

やっていること：**自動車の移動**

- **目的地の** X,Y **座標を入力**
- **移動** (**燃料が減る**), **現在位置を更新**
- **以上を繰り返す**

実行例

```
現在位置：(0.00, 0.00)
残り燃料：90.00リットル
移動しますか【Yes…1／No…0】：1
目的値のX座標：10
       Y座標：20
現在位置：(10.00, 20.00)
残り燃料：67.64リットル
移動しますか【Yes…1／No…0】：1
目的値のX座標：25
       Y座標：20
現在位置：(25.00, 20.00)
残り燃料：52.64リットル
移動しますか【Yes…1／No…0】：0
```

List 12-9 **概説** (2)

構造体 2 つ

```
/*=== 点の座標を表す構造体 ===*/
typedef struct  {
  double x;  /* X 座標 */
  double y;  /* Y 座標 */
} Point;

/*=== 自動車を表す構造体 ===*/
typedef struct  {
  Point  pt;    /* 現在位置 */
  double fuel;  /* 残り燃料 */
} Car;
```

List 12-9 概説 (3)

main 関数

初期位置は原点 $(0, 0)$, 燃料 90 で動作開始

```
int main(void)
{
  Car mycar = {{0.0, 0.0}, 90.0}; /*初期の位置と燃料*/
  while (1) {
    int select;
    Point dest;          /* 目的地の座標 */
    put_info(mycar);     /* 現在位置と残り燃料を表示 */
    printf("移動しますか【Yes…1／No…0】：");
    scanf("%d", &select);
    if (select != 1) break;
    printf("目的値のX座標：");  scanf("%lf", &dest.x);
    printf("       Y座標：");  scanf("%lf", &dest.y);
    if (!move(&mycar, dest))
      puts("\a燃料不足で移動できません。");
  }
  return 0;
}
```

List 12-9 概説 (4)

移動: move **関数**

```
/*--- cの指す車を目的座標destに移動 ---*/
int move(Car *c, Point dest)
{
  double d = distance_of(c->pt, dest);  /* 移動距離 */
  if (d > c->fuel)      /* 移動距離が燃料を超過 */
    return 0;           /* 移動不可 */
  c->pt = dest; /* 現在位置を更新（destに移動）*/
  c->fuel -= d; /* 燃料を更新（移動距離dの分だけ減る）*/
  return 1;             /* 移動成功 */
}
```

リッター 1 **メートルの燃費が悪い車です**

おわり

番外編の課題 1

複数天体の運行シミュレーション

- **天体は互いの重力で影響し合い移動**
- **1 秒ごとの各天体の位置と速度を表示**
- **質量, 初期位置, 初期速度, その他詳細は各自で適宜設定のこと**

```
#define HBNAME    64
struct hbody {
  char   name[HBNAME]; /*天体名*/
  double m;            /*質量[kg]*/
  double x,  y,  z;    /*位置[m]*/
  double vx, vy, vz;   /*速度[m/s]*/
};
#define NHB  3
struct hbody stars[NHB];
```

構造体を使わないと変数宣言はこんなにもおぞましい (個人の主観)

```
#define NHB  3
char *name[NHB];
double m[NHB], x[NHB], y[NHB], z[NHB], vx[NHB], vy[NHB], vz[NHB];
```