第12章　「構造体」のまとめ

12.1　用語のまとめ

教 p.268

ソート　データの集りを順番に並べ替えることを _____（sort）するという。

教 p.269

バブルソート　ソートの方法の一つで直感的でわかりやすいが、大きな配列に対しては効率が悪い。

教 p.270

データの関連性　前ページのプログラムで各学生の名前の配列も用意して同時に並べ替える。
このような方法では学生に関するデータ（体重や奨学金など）を付け加えるたびに、プログラム全体の構造を大きく書き換える必要が出てくる。

教 p.272

構造体　そこでひとまとまりのデータを集めて、新しいデータ構造（型）を定義する。このようなデータ構造を _____という。（配列は同種のデータをまとめたものであるが、構造体は異種のデータをまとめることができる。）

構造体を定義する利点は関数を定義する利点と似ている。

- データが階層化され、意味が理解しやすくなる
- 詳細を隠すことが可能になる
- 設計の変更が容易になる

構造体は次のような形式で宣言する。

```
_______ タグ名 {
    型 メンバ名;
    ...
    型 メンバ名;
};
```

例:

```
struct gstudent {
    char  name[20];
    int   height;
    float weight;
    long  schols;
};
```

構造体に与える名前（上の例では `gstudent`）を _____と呼び、構造体の構成要素（上の例では `name`, `height`, `weight`, `schols`）を _____と呼ぶ。

このような構造体を格納するための変数は

```
_________ _________ 変数名;
```

という形で宣言する。

例:

```
struct gstudent shibata;
```

構造体の宣言とその型の変数を同時に定義することも可能である。

```
struct test {
    int    x;
    long   y;
    double z;
} ta tb;
```

また、その場合タグ名を省略することも可能である。

```
struct {
    int    x;
    long   y;
    double z;
} ta tb;
```

ただし後者はその場限りの（あまり役に立たない）構造体になってしまう。

教 p.274

構造体のメンバ　構造体のメンバをアクセスするには _____を用いる。. は通常、ドット演算子と呼ぶ。

教 p.275

構造体の初期化　構造体を初期化するためには、配列と同様に、変数宣言の=の右辺に各メンバに与える初期値をコンマ（,）で区切って並べ、中かっこ{〜}で囲む。

教 p.281 など

構造体と関数　構造体を関数の引数や戻り値に使うときは、配列と違って構造体のコピーが作成される。（つぎの 2 つの例を比べよ。）

addPoint.c

```
#include <stdio.h>

struct point {
  int x;
  int y;
};

struct point addPoint(struct point p1, struct point p2) {
    p1.x += p2.x;
    p1.y += p2.y;

    return p1;
}

int main(void) {
    struct point p1 = {2, 3}, p2 = {1, 2}, p3;

    p3 = addPoint(p1, p2);

    printf("p1 = {%d, %d}¥n", p1.x, p1.y);
    printf("p3 = {%d, %d}¥n", p3.x, p3.y);

    return 0;
}
```

addPointArr.c

```
#include <stdio.h>

int *addPoint(int p1[], int p2[]) {
    p1[0] += p2[0];
    p1[1] += p2[1];

    return p1;
}

int main(void) {
    int p1[] = {2, 3}, p2[] = {1, 2};
    int *p3;

    p3 = addPoint(p1, p2);

    printf("p1 = {%d, %d}¥n", p1[0], p1[1]);
    printf("p3 = {%d, %d}¥n", p3[0], p3[1]);

    return 0;
}
```

だから、大きな構造体を使用するときはコピーが起こると効率が悪いので、構造体そのものではなく、構造体へのポインタを渡すことがよく行われる。

教 p.276

構造体のメンバ（->演算子）　構造体へのポインタはよく使われるので、(*ptr).mem の代わりに、ptr->mem と書くことができる。
-> はアロー（矢印）演算子と呼ばれる。

教 p.157
p.278

構造体と typedef　typedef 宣言は長い型に対して別名をつける。

```
_________ 型 新しい型名 ;
```

例:

```
struct gstudent {
    char  name[20];
    int   height;
    float weight;
    long  schols;
};

typedef struct gstudent student;
```

以降は struct gstudent の代わりに student と書くことができる。さらに一気に、

```
typedef struct {
    char  name[20];
    int   height;
    float weight;
    long  schols;
} student;
```

とすることも可能である。

教 p.280

集成体型　配列と異なり、構造体の代入は可能である。（コピーされるので、効率のため一般的にはサイズの大きな構造体は代入を避ける。）

```
int a1[] = {1, 2}, a2[2];
a2 = a1;  /* できない */

struct point p1 = {1, 2}, p2;
p2 = p1;  /* OK */
```

教 p.280

名前空間　(1) ラベル名（説明略）, (2) （構造体の）タグ名, (3) メンバ名, (4) 識別子（変数名や関数名）は、それぞれ別物なので同じ名前を使っても構わない（全くの別物として扱われる）。

教 p.281

構造体を返す関数　配列の場合と異なり、関数内で宣言した構造体を戻り値にしてもよい。

教 p.282

構造体の配列　構造体の配列を作ることももちろん可能である。

この章の冒頭のプログラム List 12-2 は構造体を使って、List 12-8 のように書き直すことができる。

こうしておくと学生のデータに変更があったとき（例えば構造体に修得単位数をあらわすメンバを追加する）も最小限の変更で済む。

教 p.284

日付と時間を表す構造体　time_t time(time_t *timep) 関数は現在の時刻を求める。struct tm *localtime(const time_t *timep) は、time_t 型を使いやすい struct tm 型に変換する。localtime の戻り値は大域変数へのポインタである。

教 p.286

メンバとしての構造体　構造体のメンバがまた構造体型（あるいは配列型など）であってもよい。