[プログラミング入門Ｉ　１０回目　２００３・１０・３０]　構造体

構造体(structure)とは

構造体とは、プログラムを論理的にわかりやすく書くための機能です。プログラムとは、コンピュータに何かをさせるための指示を与えるという役割がありますが、人間のプログラマにとってはコンピュータの中で何かをさせたい場合にそれを表現するための道具でもあります。例えば、成績表を集計するプログラムを考えてみましょう。人間がやるのであれば表という形で表現するでしょう。算数、国語、理科と３科目とすると、人数を縦、３科目の点数を横にとる表をつくります。これをプログラムで表現するには２次元配列をつくります。人数を５０人とすると、

```
int seiseki_hyou[50][3];
```

となります。これで、例えば、１０番の人の国語の成績を参照するには seiseki_hyou[9][1]となるわけです。初めの列は算数、次の列は国語、その次は理科とするわけです。つまり、算数、国語、理科を数字に割り当てるのですが、プログラムでは数字になっていますので、他人が見たときにはわかりにくくなってしまいます。自分でも後になって、プログラムをみたときに思い出すのに時間がかかってしまうということもありえます。このような時に使うのが構造体です。次のように、算数と国語、理科の成績をひとまとまりにしてデータ（の型）として名前をつけておくことができます。

```
struct seiseki {
    int sansu;
    int kokugo;
    int rika;
};
```

成績というデータは、3 つの kokugo, sansuu, rika の３つの値であると定義するわけです。これを使えば、表を次のように宣言できます。

```
struct seiseki seiseki_kyou[50];
```

これを使えば、１０番目の国語の成績は、seiseki_hyou[９].kokugo と表現することができます。

次に、住所録を管理するプログラムを考えてみることにしましょう。これも住所録を作るデータを火とまとまりにして、構造体を定義すると便利です。

```
struct personal_record {
   char name[10];  /* 名前をいれる  */
   int age:        /* 年齢 */
   char address[100]; /* 住所  */
   int tel[11];      /* 電話番号  */
};
```

このように、整数だけでなく、いろいろなデータをひとまとまりに構造体として定義することができます。構造体の中の要素のデータの名前を**フィールド(field)**、または**メンバー(member)**と言います。構造体の定義は、以下のように宣言します。

```
struct 構造体の名前　{
   フィールドのデータ型　フィールド名；
   フィールドのデータ型　フィールド名；
    …
};
```

構造体の「構造」とはデータ構造の「構造」です。プログラムでデータを論理的に扱うためにどのようなデータをひとまとまりにすればわかりやすいかを考えることがデータの構造を考えることなのです。

構造体の変数、配列、参照

構造体の宣言（定義）はデータの形だけを定義するものです。最初に配列の例を挙げてしまいましたが、構造体のデータ型をもつ変数を宣言するには、以下のように宣言します。

```
struct 構造体の名前　変数名;
```

例えば、２次元座標上の点をあらわすには次のような構造体を宣言します。

```
struct point { double x, y; }
```

フィールド x と y が同じデータ型の時には上のように一緒に宣言できます。もちろん、別々に宣言してもＯＫです。これを使って、座標上の点をあらわす変数 p は以下のように宣言します。

```
struct point p;
```

また、構造体の宣言と、変数の宣言を一緒にすることもできます。

```
struct point { double x,y } p;
```

さて、メンバーを参照するには、”.”を使います。

構造体への参照．メンバー名

です。例えば、pの x 座標を参照するには、

```
t = p.x;
```

この参照は、代入側にも使えます。そのときには、メンバーへの代入になります。

```
p.x = 199;
```

最初の例で示したとおり、構造体の配列の宣言は、

struct 構造体の名前　配列名[サイズ];

です。例えば、１０個の座標の配列は以下のようになります。

```
struct point points[10];
```

参照は、例えば、points[1].x とすればいいわけです。

データ型とは（再び）、typedef 宣言

int や **double**, **char**, **float** などは基本的な数値に関するデータ型を**基本データ型（basic data type）**といいます。また、**int ***は「整数型データへのポインタ」というデータ型です。これについては、前回説明しました。さらに言えば、任意のデータ型 **T** に対して、**T ***は「データ型**T**に対するポインタ」のデータ型です。**struct 構造体名**もデータ型です。したがって、struct 構造体のデータ型を持つデータへのポインタは、**struct 構造体 *** です。この意味については、次回説明することにして、データ型Tを持つ変数や配列は、

T　変数名;　または　T 配列名[サイズ];

と宣言しますので、構造体の変数や配列は、

struct 構造体名　変数名;　または　struct 構造体名　配列名[サイズ];

と宣言するわけです。

データ型には、**typedef 宣言**を使って、名前をつけることができます。データ型**T**に対して名前をつけるには、以下の宣言します。

typedef T データ型名;

例えば、前の座標のデータ型に名前 point_t という名前をつけるには以下のように宣言します。

```
typedef struct point { double x,y } point_t;
```

こうしておけば、座標の変数や配列は以下のように宣言できます。(_t はタイプの t)

```
point_t p;  point_t points[10];
```

文字列は、文字へのポインタでしたね。文字列のデータ型 string_t を宣言するには、以下の宣言をします。

```
typedef char * string_t;
```

あるデータ型に対して、適当な名前をつけておけば、プログラムがわかりやすくなり、書きやすくなります。プログラミングとは、やりたいことを論理的にわかりやすく表現することでもあるのです。