構造体

平成 27 年 9 月 29 日

1　構造体の意味

プログラム中の変数で複数の要素からなるデータを複数記憶する場合、今までであれば、各要素を配列で宣言することで、記憶していた。

```
/* 構造体を使わない「だっせー」宣言 */
/* 名前と国語、算数、理科のデータ */
#define SIZE 50
char name[ SIZE ][ 20 ] ;
int  kokugo[ SIZE ] ;
int  sansu[ SIZE ] ;
int  rika[ SIZE ] ;
```

しかし、この様な方法では、各要素が１セットで (１人分×人数) のデータを記憶していることが、プログラムから判断しづらい。またソートの際に、１人分のデータを一時的に覚える時に、どの様な変数を一時的に宣言するのか、いちいち判断しなくてはいけない。

やはり、複数要素のデータにて１セットのデータ構造ができていることを記載する文法があると便利となる。このための文法が**構造体**である。

2　構造体の宣言

構造体の宣言での文法は、

```
/* 構造体の宣言の文法 */
struct タグ名 {
    型１   要素名１ ;
    型２   要素名２ ;
     :
} 構造体変数名 ;
/* (1) 構造体の宣言と、構造体変数宣言を同時にする場合 */

struct タグ名 構造体変数名１ , 構造体変数名２ , ...;
/* (2) 既に宣言済みの構造体で、構造体変数を宣言する */
```

にて、複数要素からなる構造体変数を宣言できる。そして、複数の要素の一群に対し、「タグ名」の名前を付けることができる。

通常は、宣言した構造体を、複数の異なる場所で利用することが多いため、一般的に上記 (1) の様に構造体宣言と、構造体変数宣言を同時にすることは少ない。
下記の例の様に、構造体の宣言と、構造体変数宣言は、別に行うのが普通。

```
/* 構造体の宣言の例 */
struct BirthDay {
    int       year ;
    short int month ;
    short int day ;
} ;
struct BirthDay saitoh ;
struct BirthDay mitsuki ;
saitoh.year  = 1965 ; saitoh.month  = 2 ; saitoh.day  =  7 ;
mitsuki.year = 1999 ; mitsuki.month = 7 ; mitsuki.day = 14 ;
```

構造体の宣言でも、スコープが有効なので、以下の例であれば、構造体の宣言は、関数の外の一番最初に記述すべき。

```
int foo() {
   struct AB {
      int a , b ;
   } ;
   /* struct A は foo() の内部でのみ有効 */
}
void main() {
   struct AB  ab ; /* 「struct AB」って何？ */
   ab.a = 123 ;
}
```

3 構造体の要素の参照

構造体にて宣言された変数の一部分を参照する場合には、ピリオドに続いて要素名を記入する。

構造体変数名 . 要素名

にて参照する。

```
/* 構造体の参照の例 */
struct BirthDay saitoh ;
struct BirthDay table[ 10 ] ;

scanf( "%d %hd %hd" ,
       &(saitoh.year) , &(saitoh.month) , &(saitoh.day) ) ;
printf( "%d %d %d\n" ,
        table[ 0 ].year , table[ 0 ].month , table[ 0 ].day ) ;
```

なお、上記の例での アドレス取得演算子&は、構造体要素参照演算子のピリオドより、**演算子の優先順位**が低いので、&saitoh.year の様に () を省略するのが普通。

4 構造体の初期化

構造体を初期化する場合は、ほぼ配列の初期化と同じ様な記載が可能である。

```
/* 構造体の初期化の例 */
struct BirthDay saitoh = { 1965 , 2 , 7 } ;
struct BirthDay myFamily[ 3 ] = {
    { 1965 , 2 ,  7 } ,
    { 1976 , 3 , 13 } ,
    { 1999 , 7 , 14 } ,
} ;
```

5 構造体のネスト

プログラムの世界で、論理構造の中に別の論理構造がある場合、**ネスト (入れ子) 構造**と呼ばれる。

構造体でも入れ子構造にて、利用できる。

```
/* 入れ子になった構造体の例 */
struct Person {
    char             name[ 20 ] ;
    struct BirthDay  birth_day ;
} ;
struct Person saitoh ;
strcpy( saitoh.name , "斉藤 徹" ) ;
saitoh.birth_day.year  = 1965 ;
saitoh.birth_day.month = 2 ;
saitoh.birth_day.day   = 7 ;
```

6 構造体の全要素参照

構造体変数名同士で代入を行うと、構造体のデータの全要素をコピーすることができる。

```
/* 構造体要素の全代入の例 */
struct BirthDay saitoh = { 1965 , 2 , 7 } ;
void main() {
    struct BirthDay bday ;
    bday = saitoh ;
    printf( "%d %d  %d\n" , bday.year , bday.month , bday.day ) ;
}
```

ただし全要素の代入は、同じ構造体同士でのみ可能である。

```
/* 異なる構造体では全代入が失敗する例 */
struct BirthDay saitoh = { 1965 , 2 , 7 } ;
void main() {
    struct Tanjobi {
        int   year ;
        int   month ;
        int   day ;
    } tday ;
    tday = saitoh ;  /* 文法エラー */
}
```

この様な構造体全代入の文法は、初期の C 言語 (K&R C) では、文法エラーになるので注意が必要。

```
/* K&R-C で全代入をする方法 */
struct BirthDay saitoh = { 1965 , 2 , 7 } ;
void main() {
    struct BirthDay bday ;
    memcpy( &bday , &saitoh , sizeof( struct BirthDay ) ) ;
}
```

7　構造体のアドレス渡し

構造体のデータを関数にて処理する場合、関数に構造体のデータを引き渡す必要がある。この場合にはアドレス渡しを行うのが一般的である。

> 巨大な要素を持つ構造体があった場合、『値渡し』を行うと、局所変数に要素をコピーするため、場合によってはメモリを無駄に消費し、要素をコピーするのに、長時間の計算時間を要するかもしれない。
>
> 同じ理由から、配列のデータも通常は配列の先頭アドレスを関数に引き渡す『アドレス渡し』が行われる。

```
/* 構造体のアドレス渡しのプログラム例 */
void print_birthDay( struct Birthday *pday )
{
    printf( "%d %d %d" ,
            (*pday).year , (*pday).month , (*pday).day ) ;
}
int scan_birthDay( struct BirthDay *pday )
{
    return scanf( "%d %hd %hd" ,
                  &((*pday).year) , &((*pday).month) , &((*pday).day) ) ;
}
void main() {
    struct BirthDay  birth ;
    scan_birthDay( &birth ) ;
    print_birthDay( &birth ) ;
}
```

上記の例の様に、構造体 (データ構造) 毎に、「そのデータ構造を扱う」部品となる関数を作っておくと、プログラムの呼び出し側が、分かり易くなる。この様な考え方は、**オブジェクト指向**と呼ばれるプログラム開発方法に発展している。

なお、上記の例の様に、

(*構造体ポインタ変数). 要素名

といった書き方は、括弧,*, ピリオドと演算子が分かり難いので、すっきりと記載するための文法 (**アロー演算子**->) が用意されている。

構造体ポインタ変数->要素名

上記のプログラム例であれば、

```
/* アロー演算子にて記載した関数の例 */
void print_birthDay( struct Birthday *pday )
{
    printf( "%d %d %d" ,
            pday->year , pday->month , pday->day ) ;
}
```