6　構造体・共用体・列挙体

本節では基本的な型に基づいて作られた新しいデータ型を扱う。配列は同一の型の複数のデータをまとめて扱うのに対し、異なる型のデータをまとめて扱えうのが構造体である。構造体は複数の値を記憶できるのに対し、1 つの値しか記憶できないのが共用体である。配列型、構造体型、共用体型、関数型、ポインタ型は派生型 (derived type) といわれる。

また、複数の整数定数値に名前をつけたものが列挙体である。

6.1　構造体

構造体 (structure) はいくつかの要素を 1 つの名前でまとめたものである。名前はタグ (tag)、要素はメンバ (member) と呼ばれ名前がつけられる。メンバは並べられた順にメモリに割り付けられる。構造体の宣言は次のように行う。

```
struct タグ{
  型1 メンバ1;
  型2 メンバ2;
  ...
};
```

例 6.1 健康診断結果の構造体を考える。

```
struct health{
  char *name;     /* 氏名 */
  double height;  /* 身長 */
  double weight;  /* 体重 */
};
```

health が構造体のタグ名、メンバは name、height、weight である。これだけでは、struct health 型を作っただけでメモリは割り当てない。

3 人の健康診断を扱うときは、(main() 関数の前に) 構造体 struct health を宣言し、main() 関数の変数宣言のところで、

```
   struct health h1,h2,h3;
```

とする。h1、h2、h3 は struct health 型変数である。これにより、h1、h2、h3 がメモリに割り当てられる。

構造体 struct health を main() 関数でのみ利用するときは、main() 関数の変数宣言のところで、

```
struct health{
  char *name;     /* 氏名 */
  double height;  /* 身長 */
  double weight;  /* 体重 */
} h1,h2,h3;
```

と書くこともできる。

6.1.1 メンバアクセス演算子 — その 1

構造体のメンバにアクセスするにはメンバアクセス演算子”.”(ドット) を用いる。
例 6.1 の変数 h1 に値を割り当てるには、

```
h1.name="東京子";
h1.height=1.655;
h1.weight=49.3;
```

のようにする。

例 6.2 構造体 health にデータを入力し、BMI(体格指数) を求め、判定を行うプログラム。BMI は、体重 (kg) を身長 (m) の 2 乗で割った数である。

```
/* bmi-struct.c */
#include <stdio.h>
struct health{
  char *name; /* 氏名 */
  double height; /* 身長 */
  double weight; /* 体重 */
};

int main(void)
{
  double bmi;
  struct health h; /* 構造体変数の宣言 */

  h.name = malloc(100 * sizeof(char));

  printf("氏名を入力してください。");
  scanf("%s",h.name);
  printf("%s の身長 (m) を入力してください。",h.name);
  scanf("%lf",&h.height);
  printf("%s の体重 (kg) を入力してください。",h.name);
  scanf("%lf",&h.weight);

  bmi=h.weight/(h.height*h.height);

  printf("BMI(体格指数) は%f です。\n",bmi);
  printf("判定は");
  if (bmi<=17.5)
    printf("やせすぎです。\n");
  else if(bmi<=19.7)
    printf("やせ気味です。\n");
  else if(bmi<=24.1)
    printf("標準です。\n");
  else if(bmi<=26.3)
    printf("ふとり気味です。\n");
  else
    printf("肥満です。\n");

  free(h.name);

  return 0;
}
```

実行結果は以下。

```
~/comp3a$ cc bmi-struct.c -o bmi-struct
~/comp3a$ ./bmi-struct
氏名を入力してください。東京子
東京子の身長 (m) を入力してください。1.655
東京子の体重 (kg) を入力してください。49.3
BMI(体格指数) は 17.999106 です。
判定はやせ気味です。
```

6.1.2 typedef

typedef というキーワードはデータ型に新しい名前を定義する。

構造体変数の宣言は、例 6.2 では struct health h のようになるが、typedef を用いると構造体変数を宣言するときには struct をつけなくて済むようになる。

例 6.2 は次のように書ける。

```
#include <stdio.h>
typedef struct{
  char *name; /* 氏名 */
  double height; /* 身長 */
  double weight; /* 体重 */
} health;

int main(void)
{
  double bmi;
  health h; /* 構造体変数の宣言 */
(以下略)
```

typedef を使うときはタグ名はなくてよい。タグ名を使って型を次のように定義することもできる。

```
#include <stdio.h>
struct health{
  char *name; /* 氏名 */
  double height; /* 身長 */
  double weight; /* 体重 */
};
typedef struct health health; /* struct health 型の型 health を定義 */

int main(void)
{
  double bmi;
  health h; /* 構造体変数の宣言 */
(以下略)
```

演習 6.1 bmi-struct.c を typedef を使って書き直せ。

typedef は新たに定義した型を基本型と同じように使えるので便利である。

ANSI C には複素数型はないが、

```
typedef struct{
  double re; /* 実部 */
  double im; /* 虚部 */
} complex;
```

と型 struct complex を定義し、main() 関数の中などで変数を

```
complex z1,z2;
```

と宣言すれば複素数型変数 z1,z2 が使える。

6.1.3 構造体の初期化と代入

構造体変数の初期化は配列の初期化と類似の方法で行える。

```
構造体型名 構造体変数名={ 値の並び }
```

配列を配列に代入することはできないが、構造体の場合は可能である。構造体の代入は、メンバを 1 つずつコピーしたことになる。

```
/* struct-initialization.c */
#include <stdio.h>
typedef struct{
  char *name; /* 氏名 */
  double height; /* 身長 */
  double weight; /* 体重 */
} health;

int main(void)
{
  health h1={"東京子", 1.655, 49.3}; /* health 型変数 h1 の初期化 */
  health h2; /* health 型変数 h2 の宣言 */

  printf("%s の身長は%fm, 体重は%fkg です。\n",h1.name,h1.height,h1.weight);

  h2=h1; /* 代入 */
  printf("%s の身長は%fm, 体重は%fkg です。\n",h2.name,h2.height,h2.weight);

  return 0;
}
```

6.1.4 構造体を引数に取る関数

構造体を関数の引数に用いることができる。実引数のメンバの値がそれぞれコピーされて関数に渡される。つまり値渡しであって、実引数の値は変らない。

struct-initialization.c の printf 文を関数にする

```
/* struct-initialization-func.c */
#include <stdio.h>
typedef struct{
  char *name; /* 氏名 */
  double height; /* 身長 */
  double weight; /* 体重 */
} health;
void show(health h); /* 関数 show() のプロトタイプ宣言 */
```

```
int main(void)
{
  double bmi;
  health h1={"東京子", 1.655, 49.3};
  health h2;

  show(h1); /* 構造体 h1 の値を関数 show() に渡す */

  h2=h1; /* 代入 */
  show(h2);

  return 0;
}

void show(health h)
{
  printf("%s の身長は%fm, 体重は%fkg です。\n", h.name, h.height, h.weight);
}
```

演習 6.2 bmi-struct.c の入力部分を関数にせよ。(返却値は health 型である。)

演習 6.3 複素数 z1、z2 を引数に取り、z1+z2、z1-z2、z1×z2、z1/z2 を返却値とする関数 csum()、csub()、cprod()、cdiv() を作れ。ただし、複素数の四則は次のように定義される。

$$\mathrm{Re}(z1 \pm z2) = \mathrm{Re}(z1) \pm \mathrm{Re}(z2), \quad \mathrm{Im}(z1 \pm z2) = \mathrm{Im}(z1) \pm \mathrm{Im}(z2)$$

$$\mathrm{Re}(z1 \times z2) = \mathrm{Re}(z1) \times \mathrm{Re}(z2) - \mathrm{Im}(z1) \times \mathrm{Im}(z2), \quad \mathrm{Im}(z1 \times z2) = \mathrm{Im}(z1) \times \mathrm{Re}(z2) + \mathrm{Im}(z1) \times \mathrm{Re}(z2)$$

$$\mathrm{Re}(\frac{z1}{z2}) = \frac{\mathrm{Re}(z1) \times \mathrm{Re}(z2) + \mathrm{Im}(z1) \times \mathrm{Im}(z2)}{(\mathrm{Re}(z2))^2 + (\mathrm{Im}(z2))^2}, \quad \mathrm{Im}(\frac{z1}{z2}) = \frac{\mathrm{Re}(z2) \times \mathrm{Im}(z1) - \mathrm{Re}(z1) \times \mathrm{Im}(z2)}{(\mathrm{Re}(z2))^2 + (\mathrm{Im}(z2))^2}$$

ここで、$\mathrm{Re}(z), \mathrm{Im}(z)$ はそれぞれ複素数 z の実部と虚部である。すなわち、$z = \mathrm{Re}(z) + \mathrm{Im}(z)i$。

6.1.5 構造体へのポインタ

char 型や int 型へのポインタと同様に、構造体へのポインタを扱うことができる。

char 型や int 型へのポインタでは、指している変数にアクセスするのに間接参照演算子 * を用いるが、ポインタから構造体のメンバにアクセスするには、メンバアクセス演算子の一つであるアロー演算子 -> (マイナス、大なり) を用いる。

```
構造体へのポインタ -> 構造体メンバ
```

メンバがたくさんある構造体を関数の引数につかうと、メンバの値すべてがコピーされるので関数の呼び出しに時間がかかる。このようなときは、構造体のポインタを引数に使うことがある。この場合は参照渡しになるので、関数により構造体の値を変更することができる。

struct-initialization-func.c(p.71) の関数 show() の仮引数に構造体へのポインタ、実引数に構造体のアドレスを取るように変更する。(変換指定子も変更する。)

```
/* struct-poiter.c */
#include <stdio.h>
typedef struct{
  char *name; /* 氏名 */
  double height; /* 身長 */
  double weight; /* 体重 */
} health;

void show(health *pth); /* 関数 show() のプロトタイプ宣言 */

int main(void)
{
  double bmi;
  health h={"東京子", 1.655, 49.3};

  show(&h); /* 構造体 h のアドレスを関数 show() に渡す */

  return 0;
}

void show(health *pth) /* pth は構造体へのポインタ */
{
  printf("%s の身長は%.3fm, 体重は%.1fkg です。\n",h->name, h->height, h->weight);
}
```

演習 6.4 演習 6.3(p.72) をポインタを用いて書き直せ。

6.1.6 構造体の配列

構造体は char 型や int 型と同様に配列にすることができる。

構造体 health 型の配列を使う例をしめす。関数 show() は、仮引数に health 型の変数をとり、実引数に health 型配列の要素を渡す。

```
/* struct-array.c */
#include <stdio.h>
#define N 2 typedef struct{
  char *name; /* 氏名 */
  double height; /* 身長 */
  double weight; /* 体重 */
} health;

void show(health hl); /* 関数 show() のプロトタイプ宣言 */
```

```
int main(void)
{
  health h[N];
  int i;

  for(i=0;i<N;i++){
    printf("%d の氏名は",i);
    scanf("%s",h[i].name); /* ポインタのため & は不要 */
    printf("身長 (m) は",i);
    scanf("%lf",&h[i].height);
    printf("体重 (kg) は",i);
    scanf("%lf",&h[i].weight);
  }

  for(i=0;i<N;i++){
    show(h[i]);
  }

  return 0;
}

void show(health hl)
{
  printf("%10s %5.3f %5.1f \n",hl.name, hl.height, hl.weight);
}
```

演習 6.5 struct-array.c をポインタを用いて書き直せ。

6.2 共用体

共用体 (union) は構造体と類似の構文をもつ。

```
union タグ{
  型1 メンバ1;
  型2 メンバ2;
  ...
};
```

共用体のメンバーは同じ領域を共有している。構造体はメンバの数だけ値を記憶できる (たとえば、health 型は 3 つの値を記憶できる) が、共用体は全体で 1 つだけしか値を記憶できない。

共用体のサイズは最も大きいメンバのサイズとなる。

int 型が 4 バイトであるとして、共用体

```
union int_byte{
  int num;
  char c[4];
}
```

は、整数の 4 バイトを 1 バイトずつ 4 つに分けた操作に使える。

例 6.3 整数の上位 2 バイトと下位バイトを入れ替える。

```
/* exchange.c */
#include <stdio.h>
union int_byte{
  int i;
  char c[4];
};

int main(void)
{
  union int_byte n,m;

  printf("整数を入力してください。");
  scanf("%d",&n.i);

  m.c[0]=n.c[2];
  m.c[1]=n.c[3];
  m.c[2]=n.c[0];
  m.c[3]=n.c[1];

  printf("%d の上位 2 バイトと下位 2 バイトを入れ替えた整数は %d です。\n",n.i,m.i);

  return 0;
}
```

演習 6.6 EUC-JP では漢字は 2 バイトで保存される。漢字を 1 文字与え、上位バイトと下位バイトを入れ替えた漢字 (?) を出力するプログラムをつくれ。

6.3 列挙体

「いいえ」と「はい」を 0 と 1 で表す代わりに、NO と YES(あるいは no と yes) で表したり、曜日を 0, 1, 2, 3, 4, 5, 6 で表す代わりに、SUN, MON, TUE, WED, THE, FRI, SAT で表すのが列挙体 (enumeration) である。

```
enum タグ { メンバ 1=整数 1, メンバ 2=整数 2,···, メンバ n=整数 n}
```

「=整数」は省略することができる。「=整数 1」を省略すると「メンバ 1=0」となる。指定されてない値は最後に指定された値から順に一つずつ大きくなる。したがって、

```
enum タグ { メンバ 1, メンバ 2,···, メンバ n}
```

は、

```
enum タグ { メンバ 1=0, メンバ 2=1,···, メンバ n=n}
```

と同じである。列挙体のメンバを列挙定数 (enumeration constant) と呼ぶ。

例 6.4 良く使われる列挙体の例。

1. `enum boolean{FALSE, TRUE}`
2. `enum week{SUN, MON, TUE, WED, THU, FRI, SAT}`
3. `enum months{JAN=1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC}`

列挙体を用いる代わりに前処理指令`#define` で表すこともできる。

例 6.5 `enum boolean{FALSE, TRUE}` は、前処理指令に

```
#define FALSE 0
#define TRUE 1
```

と書くことと同じである。

例 6.6 列挙型の使用例。

```
/* week.c */
#include <stdio.h>

typedef enum week{SUN, MON, TUE, WED, THU, FRI, SAT} week;

int main(void)
{
  week w; /* 列挙型の変数宣言 */
  w=SUN; /* 列挙型の変数に値を代入 */

  switch(w){
  case SUN : printf("日曜です。\n"); break;
  case MON : printf("月曜です。\n"); break;
  case TUE : printf("火曜です。\n"); break;
  case WED : printf("水曜です。\n"); break;
  case THU : printf("木曜です。\n"); break;
  case FRI : printf("金曜です。\n"); break;
  case SAT : printf("土曜です。\n"); break;
  default:  printf("何曜か分かりません。\n"); break;
  }

  return 0;
}
```

演習 6.7 week.c をキーボードから曜日を入力するように書き直せ。

演習 6.8 week.c を typedef を用いずに表せ。

演習 6.9 week.c を列挙型の代わりに `#define SUN 0` などの前処理指令を用いて書き直せ。