15. 構造体

C 言語ではデータの集まりを 1 つの単位として扱うことができる。この単位のことを構造体といい、構造体を使う例として xy 平面上の点を表すデータ型を考えてみよう。

構造体宣言

点はその x 座標と y 座標の値を持つから、その 2 つをセットにして扱う方が自然である。

```
struct point
{
   float x;          /* x 座標 */
   float y;          /* y 座標 */
};
```

この例では上のように、構造体 point を宣言している。構造体は `struct` を使って宣言する。`struct` に続く部分を**タグ名**と呼ぶ。上の例では `point` がタグ名である。タグ名は、構造体のデータ型を識別するための名前として機能する。タグ名に続く中括弧の中には、その構造体の構成要素である**構造体メンバ**を書き並べる。各メンバは任意のデータ型が宣言できる。構造体 point の構造体メンバは、x および y 座標の値を格納する実数型の変数 `x` と `y` である。
宣言の一般的な書き方は次のようになる。

```
struct タグ名
{
        第 1 メンバの宣言;
        第 2 メンバの宣言;
              .
              .
              .
};
```

このようにして構造体宣言によって、データをグループ単位で扱えるデータ型を型宣言できる。

構造体変数

構造体変数とは、構造体のデータ型をとっている変数のことである。point の構造体の宣言が済んでいる時は、

```
struct point a,b;
```

のように変数 a と b を定義 (記憶割当てすることを定義という) できる。
構造体宣言と同時にその変数を定義する時は、

```
struct point
{
   float x;                  /* x 座標 */
   float y;                  /* y 座標 */
} a,b;
```

のようにする。

構造体への代入と参照

構造体メンバは、構造体変数名とメンバ名をピリオド (.) でつなげて参照する。

構造体変数名 . メンバ名

これによって構造体メンバに対しての参照、代入が他の変数と同様に行なえる。また、同一の構造体をもつ構造体変数であれば、各メンバごとに代入する必要はなく、b=a; といった代入文で一度に全てのメンバを代入することが可能である。したがって、データをまとまりとして操作する必要がある時には非常に簡単に記述することができる。

構造体変数の初期化

構造体変数の宣言時に各メンバに初期値を代入するには次のようにする。

```
struct point
{
   float x;                  /* x 座標 */
   float y;                  /* y 座標 */
} a = {1.0, 3.0};

struct point p[3] = {
                        {1.0, 1.0},         /* p[0] の初期化 */
                        {2.0, 9.0},         /* p[1] の初期化 */
                        {0.0, -1.0}         /* p[2] の初期化 */
};
```

[例題]　2 つの点の座標から、その 2 点を結ぶ線分の長さを求めてみよう。以下に 3 つのやり方を示す。

(1) 関数 main 内ですべて計算する場合

```
 1  #include <stdio.h>
 2  #include <math.h>
 3
 4  main()
 5  {
 6     struct point                                  /* 構造体を使って座標を表す */
 7     {
 8        float x;                                      /* x 座標 */
 9        float y;                                      /* y 座標 */
10     } a,b;
11
12     double jijouwa;
13
```

```
14     printf("input x-y coordinate ");
15     scanf("%f%f", &a.x, &a.y);                          /* 座標 a の入力 */
16     printf("input another x-y coordinate ");
17     scanf("%f%f", &b.x, &b.y);                          /* 座標 b の入力 */
18
19     jijouwa = (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
20     printf("length between (%.1f %.1f) and (%.1f %.1f) = %f\n",
21             a.x, a.y, b.x, b.y, sqrt(jijouwa));
22 }
```

(2) 長さを求める関数を作る(座標を値で渡す場合. このやり方は効率が良くない)

```
 1 #include <stdio.h>
 2 #include <math.h>
 3
 4 struct point                                        /* 構造体を使って座標を表す */
 5 {
 6     float x;                                          /* x 座標 */
 7     float y;                                          /* y 座標 */
 8 };
 9
10 double kyori (struct point, struct point);           /* プロトタイプ宣言 */
11
12 main()
13 {
14     struct point a,b;
15
16     printf("input x-y coordinate ");
17     scanf("%f%f", &a.x, &a.y);                        /* 座標 a の入力 */
18     printf("input another x-y coordinate ");
19     scanf("%f%f", &b.x, &b.y);                        /* 座標 b の入力 */
20
21     printf("length between (%.1f %.1f) and (%.1f %.1f) = %f\n",
22             a.x, a.y, b.x, b.y, kyori(a, b));
23 }
24
25 double kyori (struct point p, struct point q)
26 {
27     double jijouwa;
28
29     jijouwa = (p.x - q.x) * (p.x - q.x) + (p.y - q.y) * (p.y - q.y);
30     return(sqrt(jijouwa));
31 }
```

(3) 長さを求める関数を作る(座標をポインタで渡す場合)

```
 1  #include <stdio.h>
 2  #include <math.h>
 3
 4  struct point                                    /* 構造体を使って座標を表す */
 5  {
 6     float x;                                          /* x 座標 */
 7     float y;                                          /* y 座標 */
 8  };
 9
10  double kyori (struct point *, struct point *);       /* プロトタイプ宣言 */
11
12  main()
13  {
14     struct point a,b;
15
16     printf("input x-y coordinate ");
17     scanf("%f%f", &a.x, &a.y);                        /* 座標 a の入力 */
18     printf("input another x-y coordinate ");
19     scanf("%f%f", &b.x, &b.y);                        /* 座標 b の入力 */
20
21     printf("length between (%.1f %.1f) and (%.1f %.1f) = %lf\n",
22             a.x, a.y, b.x, b.y, kyori(&a, &b));       /* アドレスを渡す */
23  }
24
25  double kyori (struct point *p, struct point *q)
26  {
27     double jijouwa;
28
29     jijouwa = (p->x - q->x) * (p->x - q->x) + (p->y - q->y) * (p->y - q->y);
30            /* ^^^^^^^^^^^^^ の部分は ((*p).x - (*q).x) という書き方でもよい  */
31     return(sqrt(jijouwa));
32  }
```

[練習 1] 2 つの点の座標を受けとって、その 2 点の中点の座標を求める関数 middle を定義しなさい。

[練習 2] 平面上の 3 点の座標から、それで囲まれる三角形の面積を求めるプログラムを作りなさい。
(ヒント) ヘロンの公式：三角形の三辺の長さを a, b, c とすると、その面積 S は $S = \sqrt{s(s-a)(s-b)(s-c)}$ である。ただし、$2s = a + b + c$ とする。

ところで、さきほどの練習問題「2 点の中点の座標を求める問題」では、例題の解答例 (2) のように座標をポインタで渡し、座標の構造体を返してもらう解答もありうる。しかし、この方法では、関数 middle の中で作成された構造体をコピーすることで受け取っているので効率が良くない。この無駄を省くことはできないだろうか？ 一つの方法は、例題の解答例 (3) のように中点用の変数のアドレスを渡してそこに値を代入してもらえばよいだろう。では、middle 内の構造体変数をポインタで返してもらうことはできないだろうか？ 実は、関数 middle の中で作られた変数は局所変数なので、関数呼出し後に関数内の変数をアクセスすることはできず、たとえポインタとして返してもらってもそれを使うことができない。(ビルドするとおそらく「局所変数のアドレスを返値としている」といった警告が出るだろう)
では、どうしたらよいだろうか。一つの方法は、middle 関数内でその変数用の領域を特別な方法で獲得して、関数呼出し後も生き続けさせることである。その方法とは領域を動的に獲得するもので、malloc() という組込関数を使用する。malloc(x) の引数 x は、必要とするバイト数である。この場合は point 構造体に要するバイト数だが、変数や型に要するバイト数を求めるには、関数 sizeof() を使用する。

```
#include <stdio.h>
#include <malloc.h>

struct point                                        /* 構造体を使って座標を表す */
{
   float x;                                                 /* x 座標 */
   float y;                                                 /* y 座標 */
};

struct point* middle (struct point *, struct point *);  /* プロトタイプ宣言 */


main()
{
  struct point a,b, *middlepoint;

   printf("input x-y coordinate ");
   scanf("%f%f", &a.x, &a.y);                              /* 座標 a の入力 */
   printf("input another x-y coordinate ");
   scanf("%f%f", &b.x, &b.y);                              /* 座標 b の入力 */

   middlepoint = middle(&a, &b);

   printf("mid point of (%.1f %.1f) and (%.1f %.1f) = (%.1f, %.1f)\n",
    a.x, a.y, b.x, b.y, middlepoint->x, middlepoint->y);
   /* free(middlepoint); */
}


struct point* middle (struct point *p, struct point *q)
{
   struct point * midpoint;

   midpoint = (struct point *) malloc(sizeof(struct point));
         /* (struct point *) でキャストして、型をそろえてから代入 */
   midpoint->x = (p->x + q->x) / 2;
   midpoint->y = (p->y + q->y) / 2;

   return (midpoint);
}
```