

[プログラミング入門 I (11)] おまけ

構造体の代入、引数渡し、返り値、sizeof 演算子

構造体とは、いくつかのデータをひとまとまりにしたものです。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には必要な機能です。

前回の最後にデータ型について、説明しましたが、同じデータ型同士であれば、代入ができます。例えば、座標を表す point という名前の構造体定義されているものとしましょう。

```
struct point { int x,y; }; /* 構造体の定義 */
struct point A,B;
```

と宣言されていれば、以下のように代入できます。(演算はできません)

```
A = B;
```

代入とは、構造体のコピーです。

当然、引数にも使うことができます。関数 foo が以下のように定義されていれば、

```
void foo(struct point a, struct point b){ ... }
```

これで、

```
foo(A,B);
```

と書けば、呼び出しに構造体をつかうことができます。以前、関数の説明をしましたが、C 言語の場合は「値渡し」です。つまり、引数の値はコピーされて、引数として関数に渡されます。

また、関数の返り値としても返すことができます。例えば、以下のようにすることができます。

```
struct point goo(...) { struct point X; ... return X; }
```

受け取るほうは、

```
A = goo(...);
```

として、返り値をコピーすることができます。構造体は、データ型ですので、int や double と同じようにつかうことができます。struct point と書くのが面倒だったら、前回、説明したとおり、typedef をつかって、データ型を定義しておけばすこしは書く量が減って、わかりやすくすることもできます。

```
typedef struct point point_type;
point_type A,B;
```

とかけます。

以上、代入、関数の引数、返り値、どれも、構造体の場合も int や double と同じように「コピー」されます。コピーなので、大きな構造体の場合は時間がかかってしまうことになります。では、どのくらいのサイズなのか。データ型のサイズは、sizeof 演算子を使って調べることができます。例えば、int は、4 バイトなので、sizeof(int) は 4 になります。例えば、上の 2 つの整数をメンバーとしてもつ構造体 point のサイズをしらべてみましょう。

```
printf("size is %d\n",sizeof(struct point));
```

を実行してみると、4 バイトの int が 2 つなので、8 とプリントされるはずですが、

構造体とポインタ、malloc 関数

構造体を上のように、直接、引数に使ってしまうと、コピーされるため、不効率になってしまうことがあります。前の例では整数が 2 つだけなので、たいしたことはありませんが、100 バイトもある構造体の場合はいちいちコピーしていたら、遅いプログラムになってしまいます。(実際、このような小僧体の代入や引数は最初の C 言語にはない機能でした。多くのプログラムでは構造体に引数を直接かくことは注意する必要があります)

このような場合は、ポインタを使います。ポインタはプログラムを効率的なプログラムを書くための機能です。ポインタの宣言は、データ型のあとに *をつけたものですから、前の例の point 構造体へのポインタの宣言は、struct point * で宣言します。前の例の関数宣言を書き直してみましょう。

```
void foo(struct point *ap, struct point *bp){ ... }
```

呼び出しのほうはポインタを渡すわけですから、&演算子を使って、ポインタをつかって渡します。

```
foo(&A,&B);
```

ポインタからメンバーの値を参照するには、->演算子をつかいます。たとえば、ap で指されている構造体から、メンバー x を参照するには、

```
t = ap->x + 1;
```

と書きます。これは左側にかけて、メンバーへの代入になります。

```
ap->x = 123;
```

これは、まず構造体を参照して、そのメンバーを参照すればいいので、`ap->x` は、

```
(*ap).x
```

ともかくことができます。しかし、`->`のほうがわかりやすいので、こちらを使うようにしましょう。

以前の資料に書いておきましたが、メモリを確保する関数に `malloc` という関数があります。この関数は、`malloc(確保するバイト数)` で呼び出します。`malloc` 関数から返る値は、確保されたメモリのアドレス、つまりポインタです。使っているマシンの環境にもよりますが、`malloc` 関数は文字へのポインタ、つまり、`char *` で、`<stdlib.h>` で宣言されていることがおおいので、必要なメモリを取ったあとは適当なデータ型のポインタに変えてやる必要があります。これをやるのが、キャスト演算子です。

```
ap = (struct point *)malloc(sizeof(struct point));
```

必要なバイト数は `sizeof` 演算子を使って計算しています。

このように、プログラムの中で実行中にメモリを確保するようなプログラムでは構造体へのポインタは不可欠な機能です。