インタラクティブゲーム制作
ゲームプログラミング講座
第 4 回資料

竹内 亮太

(2009/6/5)

4　ポインタとクラス化

4.1　まずはおさらい

今回も引き続き、ポインタにまつわる基礎知識を整理していきます。まずは前回の宿題の 1 つを解決しておきましょう。こんな感じでどうでしょうか？

```
void swapInt(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
    return;
}
```

ちなみに、この int を double に置き換えれば、実数用の関数に早変わりです。呼び出す時には、

```
    int valueA = 5, valueB = 8;
    swapInt(&valueA, &valueB);
```

このようにして使います。「変数の場所を渡す」という発想がないと、こういう処理を関数化することはできなかったはずです。

4.2　実体のような実体でないもの

もう一つ、ポインタ以外に C++ならではの書き方があります。「参照」というシロモノです。

```
void swapInt(int &a, int &b)
{
    int t = a;
    a = b;
    b = t;
    return;
}
```

引数の型が int & となっている点に注意してください。こうした場合、関数を呼び出す際にはこうなります。

```
    int valueA = 5, valueB = 8;
    swapInt(valueA, valueB);
```

あれ、実体を渡しちゃっていいの？と思うかも知れませんが、この組み合わせと先に示した組み合わせは全く同じ動作になります。& を付けて変数を受けることで、ポインタを使った場合と同じような動作を自動的にしてくれる、というだけのことです。「同じ変数を別名で扱える」という解釈でも構いません。主に関数の引数で、ポインタを直接操作するのが嫌な場合に使います。それ以外の用途ではまず使いません。引数専用だと思っていいでしょう。

メールでも書きましたが、私個人はあまりこれを使いません。が、利用しているライブラリやサンプルプログラムも多いと思いますので、紹介はしておきます。混乱しそうなら無理に使う必要はないでしょう。

4.3　クラスオブジェクトのポインタ

そして、今までは int 型の場合の話だけしていましたが、クラスオブジェクトを扱う場合はもう一つ追加ルールがあります。

- 実体 (もしくは参照) のメンバにアクセスする時は「変数名 . メンバ」
- ポインタを通じてメンバにアクセスする時は「ポインタ名->メンバ」

なんでこうしちゃったんかなーと私もたまに思うのですが、ピリオド (.) と矢印みたいなヤツ (->) を使い分けることになっています。矢印みたいなヤツのことはその名の通り「アロー演算子」と呼んだりもします。まぁこうすることで、今扱っている変数が実体なのか、ポインタなのか、区別ができるので面倒くさいだけではないと思います。

ちなみに「(&変数名)->メンバ」とか「(*ポインタ).メンバ」なんてこともできますが、**キモイのでやめましょう。**

4.4 実体やらポインタやらのまとめ

とりあえず、ごちゃっと出てきたのでまとめます。

表 1: C++における変数関係のまとめ

宣言の仕方	CHoge a;	CHoge *a;	CHoge &a;
種類	実体	ポインタ	参照
代入物	実体・定数	アドレス	実体を必ず代入
メンバへのアクセス	.	->	.
& 付け	アドレス	ポインタのアドレス	アドレス
*付け	エラー！	実体	エラー！

多くの場合、実体とポインタの使い分けさえできれば問題はありません。参照は宣言と同時に実体の代入が必要なため、ほとんどの場合で関数の引数にしか使われません。

ポインタに & を付けると「アドレスを覚えている変数のアドレス」が手に入ります。ダブルポインタと言うこともあります。Direct3D を使ってると有無を言わさず使わされるてじょう。自分のプログラムの場合、通常の設計では使うことはないか、もしくは使う必要がありそうな場合でも、使わずに済む方法があるはずですので、設計を見直した方が良いでしょう。

4.5 自分のプログラムをクラス化する

さて、ポインタに関する知識を身に付けたところで、いよいよ自分のプログラムをクラス化していきましょう。ゲームプログラムにおいてクラス化するべきものは、

- プレイヤーキャラ
- 敵キャラ
- 背景セット
- 画面上のゲージやメーターなどの表示物
- それらをひっくるめた「画面全体」

とりあえずざっと挙げただけでもこのくらいあります。うへぇ、となりそうですが、**一気にやろうとするのが挫折のもと**です。手の動くところからコツコツ進めていきましょう。

また、最初から上手にクラス化出来る人などいません。一度クラス分けしたものでも、開発を進めていくとしっくり来なくなったりします。そういう場合はまた再設計したり見直したりしていくことで、構造が洗練されていきます。まずは今出来ることやる。これが重要です。

とりあえずはプレイヤーキャラ (といっても直方体ですが) をクラス化することを考えます。この場合、メンバ変数として持たせるべきは当然、「プレイヤーキャラを構成している変数」です。fk_Scene と切り離されてしまうように思いますが、それは後で解決します。

次にメンバ関数として何を持たせるかですが、とりあえず必殺パターンを伝授します。

- 初期化処理
- ループ 1 回分の処理
- シーンへの登録・消去処理
- キー入力処理

とりあえずはこんくらいあれば十分です。先週 Car サンプルをよーく読んだ人は「パクリじゃん？」と思うかもしれません。が、それでいいんです。最初のうちは他人のクラス構成を真似るのが手っ取り早いです。それがうまく馴染めば儲けもの、段々合わなくなってきたら、その頃には自力で設計を考えられるだけの経験値は積んでいると見て良いでしょう。

シーンへの登録処理やキー入力では、fk_Scene や fk_Window の変数が必要になります。じゃあこれらもメンバ変数に持たないといけないの？と思ってしまいがちですが、そんなことはありません。必要な時にだけ**ポインタを引数で渡す**ようにすれば、不要なメンバ変数を増やす必要はありません。Car の entryScene などはそうやって解決していますよね？確かめてみてください。

関数名も最初は真似っこでいいでしょう。クラスを設計する場合は**変数は名詞形、関数は動詞形 (+目的語)** というのがお決まりになっています。初期化処理は”init”、ループ 1 回分の処理は”update”などとするのが定番です。

まずはやってみましょう。そしてうまくできたら背景部分もクラス化してみましょう。