Polimorfismo

Esta é uma ótima definição acerca do polimorfismo:

> Considere o exemplo de polimorfismo a seguir. Suponha que criamos um programa que simula o movimento de vários tipos de animais para um estudo biológico. As classes `Peixe`, `Anfíbio` e `Pássaro` representam os três tipos de animais sob investigação. Imagine que cada uma dessas classes herda da classe básica `Animal`, que contém uma função mover e mantém a localização atual de um animal. Toda classe derivada implementa a função `mover`. [...] Para simular os movimentos dos animais, o programa envia a mesma mensagem a cada objeto uma vez por segundo – a saber, `mover`. Entretanto, cada tipo específico de `Animal` responde a uma mensagem de mover de maneira própria e única – um `Peixe` poderia nadar dois metros, um `Sapo` poderia pular três metros e um `Pássaro`, voar dez metros. O programa emite a mesma mensagem (isto é, `mover`) para casa objeto animal genericamente, mas cada objeto sabe como modificar sua posição apropriadamente de acordo com seu tipo de movimento específico. Contar com o fato de que cada objeto sabe 'fazer a coisa certa' (isto é, faz o que é apropriado a esse tipo de objeto) em resposta à mesma chamada de método é o conceito-chave do polimorfismo. A mesma mensagem (nesse caso, `mover`) enviada a uma variedade de objetos tem 'muitas formas' de resultados – daí o termo polimorfismo". (DEITEL, 2006, pg. 546)

Transcrevendo o exemplo supracitado para a linguagem C, ficaria algo tipo:

```
1.  struct Animal{
2.          char* name;
3.  };
4.
5.  struct Fish{
6.          struct Animal anFish;
7.          int toSwim;
8.  };
9.
10. struct Amphibian{
11.         struct Animal anFrog;
12.         int toJump;
13. };
14.
```

```
15. struct Bird{
16.         struct Animal anBird;
17.         int toFly;
18. };
19.
20. static int location = 0;
21.
22. int toMove(int m){
23.         location = m;
24.         return location;
25. }
```

A “classe base” `Animal` é definida na linha 1. Em seguida, são definidas as “classes derivadas” `Fish` (`Peixe`), na linha 5, `Amphibian` (`Anfíbio`), na linha 10, e `Bird` (`Pássaro`), na linha 15. Todas as “classes”/estruturas (linhas 5, 10 e 15), recebem um membro do tipo struct `Animal`, afim de referenciar esta “classe” (linha 1), e outro membro do tipo inteiro (linhas 7, 12 e 17), que definirá o quanto cada animal se moveu.

Por se tratar de uma variável estática (linha 20), C não permite que ela fique dentro das “classes”/estruturas. Uma das soluções é acessá-las, através dos objetos, com o auxílio de uma “função membro” sintetizada – que, neste caso, acabou se tornando uma “função virtual” (linha 22), dado que seu comportamento, tal qual o da variável estática, respondia de acordo com cada tipo específico da “classe”/estrutura `Animal`.

```
1.  struct Animals animal = {"Animal"};
2.          printf("Nome do animal: %s, %s se moveu %d metros\n", animal.name,
    animal.name, location);
3.
4.          struct Fish fish = {{"Peixe"}, 2};
5.          location = toMove(fish.toSwim);
6.          printf("Nome do animal: %s, %s nadou %d metros\n", fish.anFish.name,
    fish.anFish.name, location);
7.
8.          struct Amphibian frog = {{"Sapo"}, 3};
9.          location = toMove(frog.toJump);
10.         printf("Nome do animal: %s, %s pulou %d metros\n", frog.anFrog.name,
    frog.anFrog.name, location);
11.
12.         struct Bird bird = {{"Pássaro"}, 10};
13.         location = toMove(bird.toFly);
14.         printf("Nome do animal: %s, %s voou %d metros\n", bird.anBird.name,
    bird.anBird.name, location);
```

```
Nome do animal: Animal, Animal se moveu 0 metros
Nome do animal: Peixe, Peixe nadou 2 metros
Nome do animal: Sapo, Sapo pulou 3 metros
Nome do animal: Pássaro, Pássaro voou 10 metros
```

Neste trecho de código é mostrado como a variável estática, que não pôde ser declarada dentro de uma “classe”/estrutura sendo um de seus membros – apesar de o sê-lo -, é acessada, através dos objetos, com o auxílio de uma “função virtual” (linhas 5, 9 e 13).