Subprogramas

Gustavo Scaloni Vendramini
Guilherme José Henrique
Sean Carlisto de Alvarenga
Vinícius Fernandes de Jesus

23 de setembro de 2013

Sumário

1 Subprogramas Como Parâmetro

Em muitas ocasiões temos a necessidade de passar um subprograma através de um parâmetro. A ideia é interessante e simples, mas gera duas complicações em termos de implementação.

Primeiro, temos a complicação que consiste na maneira de realizar a checagem de tipo (*type checking*, **??**) do subprograma passado por parâmetro. Em C e C++, onde a passagem de subprogramas é feita através de ponteiro para função, essa checagem é feita pelo tipo do ponteiro.

A outra complicação ocorre em linguagens de programação que permitem subprogramas aninhados. O problema refere-se a qual ambiente de referência o subprograma passado por parâmetro terá. Nessa situação, há três tipos possíveis:

Shallow Binding: O ambiente é o local onde o subprograma é chamado.

Deep Binding: O ambiente refere-se onde o subprograma foi definido.

Ad Hoc Binding: O ambiente condiz com o local que o subprograma foi passado por parâmetro.

Como exemplo, considere a listagem 1, cuja sintax é de JavaScript. O subprograma *sub2()* apenas imprime o valor da variável *x*, porém, seu valor depende do ambiente de referência utilizado.

```
1  function sub1() {
2    var x;
3    function sub2() {
4      alert(x);
5    };
6    function sub3() {
7      var x;
8      x = 3;
9      sub4(sub2);
10   };
11   function sub4(subx) {
12     var x;
13     x = 4;
14     subx();
15   };
16   x = 1;
17   sub3();
18 };
```

Listing 1: Código retirado de [3]

Caso a listagem em questão utilize o ambiente de referência *Shallow Binding*, o valor impresso seria 4. Caso o ambiente *Deep Binding* fosse escolhido, o valor impresso seria 1. Já para *Ad Hoc Binding*, o valor seria 3.

Segundo Robert W. Sebesta, a abordagem *Ad Hoc Binding* nunca foi implementada [3].

2 Chamar Subprogramas Indiretamente

Há momentos, durante a programação de um software, em que se torna necessário chamar subprogramas de forma indireta. Isso ocorre quando o subprograma a ser chamado é somente conhecido em tempo de execução, como eventos disparados por bibliotecas de interface gráfica e funções de *callback*.

Em C e C++, podemos utilizar ponteiro para função para chamar um subprograma conhecido em tempo de execução [1, 3]. Para utilizar essa técnica, primeiro temos que declarar uma função, como por exemplo:

```
int sum(int a, int b)
{
    return a + b;
}
```

A seguir, temos que declarar um ponteiro com a mesma assinatura da função escolhida. Como no nosso exemplo (função *sum*) a função possui dois parâmetros e um retorno do tipo *int*, temos um ponteiro como mostrado:

```
int (*sum_pointer)(int, int);
```

Em seguida, é necessário atribuir a função em questão para o ponteiro declarado. Em nosso exemplo:

```
sum_pointer = &sum;
```

Por fim, basta invocar a função, como da seguinte maneira:

```
(*sum_pointer)(1,2);
```

Em C#, podemos referenciar métodos em forma de objetos, através do uso de *delegate* [2, 3], o que torna muito poderoso e flexível. Para fazer uso do mesmo, precisamos declarar um *delegate* para um determinado protocolo de função, como:

```
public delegate int SumDelegate(int a, int b);
```

Podemos instanciar um SumDelegate passando por parâmetro para seu construtor o nome de uma função cuja declaração tenha o mesmo protocolo. Suponha a existência da função chamada *sum* que respeite essa assinatura, podemos então instanciar SumDelegate como segue:

```
SumDelegate sumDelegate= new SumDelegate(sum);
```

Para executar o *delegate* em questão, fazemos da seguinte forma:

```
sumDelegate(2,3);
```

3 Sobrecarga de Subprogramas

A maioria das linguagens de progamação permitem sobrecarga de subprogramas, que consiste em subprogramas com o mesmo nome, mas com parâmetros diferentes, seja por quantidade, ordem ou tipo.

Essa técnica é utilizada quando temos subprogramas que tem o mesmo objetivo, porém fazem uso de parâmetros diferentes. Em C++, C# e Java a sobrecarga de construtores é a mais utilizada por desenvolvedores [4].

Conforme apontado por Sebesta, as linguagens Ada, Java, C++, C# e F# são exemplo de linguagens que permitem sobrecarga de operadores [3].

Referências

[1] Alex Allain. Function pointers in c and c++. `http://www.cprogramming.com/tutorial/function-pointers.html`. Acessado em: 22/09/2013.

[2] Microsoft Developer Network. Delegates (c# programming guide). `http://msdn.microsoft.com/en-us/library/ms173171(v=vs.90).aspx`. Acessado em: 22/09/2013.

[3] Robert W. Sebesta. *Concepts of Programming Languages (10th Edition)*. Addison-Wesley, 2012.

[4] The Java Tutorials. Defining methods. `http://docs.oracle.com/javase/tutorial/java/javaOO/methods.html`. Acessado em: 23/09/2013.