Universidade Federal do Rio Grande do Sul – UFRGS

Disciplina: Modelos de Linguagem de Programação

Prof. Dr. Jéferson Campos Nobre

Prova P2

Aluna: Letícia dos Santos

Data: 26/11/2020

---

1. (2,0) Explique os cinco tipos de passagem de parâmetros estudados em aula: por valor, por resultado, por valor-resultado, por referência e por nome. Use exemplos. Identifique a quais modos tais tipos pertencem

Existem os modos de:

- **Entrada:** valor passado para a chamada, variável original não é modificada.
- **Saída:** a função retorna um valor computado no parâmetro informado.
- **Entrada e saída:** junção dos modos já apresentados. Valor retornado na variável original.

A passagem de parâmetros pode ser:

- **Por valor:** o valor do parâmetro inicializa o parâmetro formal correspondente, que será utilizado como uma variável local na função. Utiliza modo de entrada. Exemplo em C:

```
void Imprime_X(int max){ for(int i=0; i<max; i++) printf("X\n"); }
```

- **Por resultado:** o valor resultante é colocado na variável passada como parâmetro. Utiliza modo de saída. Exemplo em C#:

```
void novo_valor(out int x){ x=2020; }
```

- **Por valor-resultado:** junção dos tipos já apresentados. Também chamado de passagem por cópia, pois o parâmetro real é copiado para o formal na entrada da função e então copiado de volta no término. Utiliza modo de entrada e saída. Exemplo em C#:

```
void add4(out int x){ x+=4; }
```

- **Por referência:** transmissão de um caminho de acesso, utilização de ponteiros. Utiliza modo de entrada e saída. Exemplo em C++:

```
void funcao(int &valor){...}
```

- **Por nome:** o parâmetro real é textualmente substituído pelo parâmetro formal correspondente em todas as suas ocorrências na função. Utiliza modo de entrada e saída. Exemplo para T em C#:

  namespace Cshp_Generico{ public class Executa<T>{private T valor;} }

2. (1,5) Discuta escopo estático e dinâmico, descrevendo uma forma de implementar cada um deles. Considere vantagens e desvantagens.

Escopo estático:

Possui esse nome pois o escopo de uma variável pode ser determinado estaticamente, ou seja, antes da execução. Assim, também é possível determinar o tipo. A **implementação** pode ser feita utilizando uma lista e o registro do pai estático do escopo: todas as variáveis declaradas no escopo são colocadas em uma lista durante a compilação. Em tempo de execução, se a variável referenciada não estiver na lista, busca na lista do pai estático e assim sucessivamente até encontrar a declaração da variável.

O fato de conhecer o tipo do dado antes da execução pode ser uma vantagem, pois o compilador pode identificar erros com mais facilidade e o programa possui um comportamento determinístico, ou uma desvantagem, oferece pouca flexibilidade. Já o fato dos escopos que podem ser aninhados, ou seja, a variável utilizada pode estar declarada no pai estático, dificulta a legibilidade e torna a busca da declaração mais complexa.

Escopo dinâmico:

Baseado na sequência de chamadas de subprogramas, não em seu relacionamento espacial uns com os outros. Ou seja, o escopo pode ser determinado apenas durante a execução. Então para a **implementação** pode ser utilizada uma pilha e o registro do pai dinâmico, que serão construídos durante a execução. É necessária a estrutura de pilha porque não é conhecido o tipo das variáveis e a ordem de chamamento dos escopos também influencia a manipulação dos dados.

A flexibilidade desse tipo de escopo pode ser uma vantagem, facilita a reutilização de códigos e produz programas mais genéricos, ou uma desvantagem, o comportamento do programa é menos previsível e a legibilidade é prejudicada.

3. (1,5) Discuta tipagem dinâmica e estática. Neste contexto, relacione o conceito de *duck typing*. Quais são os impactos esperados em relação à segurança (safety) da linguagem?

Tipagem estática: determinação e verificação de tipos durante a compilação. Se não forem encontrados erros durante a compilação, é garantido que o programa é seguro (safety) para as propriedades de tipo de todas as possibilidades de entrada.

Tipagem dinâmica: determinação e verificação de tipos durante a execução. Podem acontecer erros de tipo durante a execução, o que compromete a segurança (safety). Algumas linguagens que utilizam esse sistema possuem uma forma de antecipar a falha e se recuperar, como a estrutura try em Java.

Duck typing: se anda como um pato e faz barulho como um pato, então deve ser um pato. Um objeto é definido pela presença de métodos e propriedades em detrimento do tipo em si. É um processo dinâmico. Quando comparado com tipagem estática, é semelhante a compatibilidade de tipos e equivalência de estrutura.

4. (1,0) Diferencie avaliação ansiosa (*eager evaluation*) e avaliação tardia/adiada (*lazy/delayed evaluation*). Relacione sua resposta com os paradigmas procedurais e funcionais.

Eager evaluation: a expressão é avaliada na primeira vez que é encontrada e seu resultado é vinculado a uma variável. Essa estratégia é utilizada em linguagens do paradigma procedural. Facilita a legibilidade do código e previsibilidade de funcionamento.

Lazy evaluation: atrasa a avaliação da expressão até o seu valor ser necessário e evita que a mesma avaliação seja feita mais de uma vez. Isso permite definir estruturas potencialmente infinitas, como um vetor de naturais em uma linguagem funcional. Se aproxima de conceitos matemáticos, assim como o paradigma funcional, pois quando é definido o vetor já citado não significa que necessariamente serão utilizados e avaliados todos os elementos do vetor.

5. (2,0) Conforme apresentado em aula, duas abordagens principais para garbage collection podem ser definidas: *reference counting* e *tracing*. Descreva essas abordagens e cite um exemplo de estratégia de implementação para cada uma. Cite linguagens de programação que usam tais abordagens/estratégias de implementação.

Reference counting: conta o número de referências que apontam para a célula de memória. **Usado em python.** Pode ser implementado adicionando um contador a cada célula que será incrementado quando uma nova referência é feita e decrementado quando uma referência é liberada. Quando o contador for zero, a célula de memória é liberada.

Tracing: guarda o caminho de referências entre as células de memória. **Usado em Ruby.** Pode ser implementado com uma flag indicando se a célula faz referência a outra. É necessária a fase de marcar: todas as células alcançadas a partir da raiz são marcadas. Após, as células com a flag desligada serão liberadas durante a fase de varredura.

6. (2,0) Em linguagens Orientadas a Objetos modernas, existem pelo menos quatro tipos de polimorfismo: polimorfismo *ad hoc* por coerção; polimorfismo por sobrecarga; polimorfismo universal paramétrico; e polimorfismo universal por inclusão. Explique, diferencie e dê exemplos.

Polimorfismo é quando uma única interface pode ser utilizada para entidades de tipos diferentes ou único símbolo para representar tipos diferentes.

Existem pelo menos quatro tipos:

- **Ad hoc:** mesma interface.
    - **Por coerção:** funções que podem ser aplicadas em tipos de dados diferentes quando os tipos são compatíveis no contexto do programa. Acontece uma conversão implícita de tipos nos parâmetros. Exemplo:
        - Aluno é subclasse de Pessoa. Então, a função Obter_nome também pode ser aplicada sobre um Aluno.
    - **Por sobrecarga:** o mesmo identificador pode representar subprogramas diferentes e o contexto ou os tipos dos parâmetros passados definem qual será selecionado. Exemplo:
        - int Soma(int a, int b){…} e float Soma(float a, float b){…}
- **Universal:** mesmo símbolo.
    - **Paramétrico:** Possibilita escrever funções genéricas que podem utilizar diferentes tipos. Exemplo em C++ utilizando template <typename T>:
        - T Soma(T a, T b){ return a+b; }
    - **Por inclusão:** um tipo de dado está dentro de outro. Relação de classe e subclasse que permite, por exemplo, que um ponteiro da classe pode apontar para uma subclasse. Exemplo utilizando Pessoa e Aluno:
        - Pessoa fulano = new Aluno();

CORREÇÃO:

Q1:2,0

Q2:1,5

Q3:1,5

Q4:1,0

Q5:1,25 - Explicar mais.

Q6:2,0