Universidade Federal do Rio Grande do Sul – UFRGS

Disciplina: Modelos de Linguagens de Programação

Prof. Dr. Jéferson Campos Nobre

Questionário P1

1. Discuta as características de linguagens de programação a seguir: simplicidade, ortogonalidade e expressividade. Como elas impactam (positivamente e negativamente) nas propriedades legibilidade, escritabilidade e confiabilidade.

TABELA 1.1 Critérios de avaliação de linguagens e as características que os afetam

	CRITÉRIOS		
Característica	**Legibilidade**	**Facilidade de escrita**	**Confiabilidade**
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipos			•
Tratamento de exceções			•
Apelidos restritos			•

   - Simplicidade: uma linguagem com muitas construções básicas é mais difícil de aprender. Mesmo para multiplicidade de recursos (várias formas de fazer a mesma operação) e sobrecarga de operadores(+ pode tanto fazer soma de int e float quanto concatenar vetores ou somar todos os elementos do vetor). Por outro lado, a simplicidade extrema do assembly prejudica a legibilidade. A simplicidade melhora a legibilidade, escritabilidade e confiabilidade.
   - Ortogonalidade: um conjunto pequeno de construções primitivas pode ser combinado a um conjunto pequeno de formas para

construir as estruturas de controle e de dados da linguagem. Simetria de relacionamento entre as primitivas. A falta leva exceções às regras da linguagem, menos simplicidade, legibilidade e escritabilidade. Excesso leva a uma complexidade desnecessária.

- Expressividade: Medida de naturalidade com que uma estratégia de resolução do problema pode ser transformada em uma estrutura de programa.

2. Diferencie os paradigmas imperativo e declarativo. Cite e explique um (sub)modelo em cada um dos paradigmas. Quais são os formalismos computacionais considerados? Fonte: Wikipédia
    - **Paradigma Imperativo:** focuses on describing **how a program operates.** This paradigm uses statements that change a program's state. In much the same way that the imperative mood in natural languages expresses commands, an imperative program consists of commands for the computer to perform.
        i. Procedural programming is a type of imperative programming in which the program is built from **one or more procedures (also termed subroutines or functions).**
    - **Paradigma Declarativo:** focuses on **what the program should accomplish** without specifying how the program should achieve the result. Makefiles, for example, specify dependencies in a declarative fashion,[6] but include an imperative list of actions to take as well. Similarly, yacc specifies a context free grammar declaratively, but includes code snippets from a host language, which is usually imperative (such as C).
        i. Many markup languages, such as **HTML**, are often declarative. HTML, for example, only describes what should appear on a webpage - it specifies neither control

flow for rendering a page nor the page's possible interactions with a user.

3. Cite e explique quatro mecanismos importantes do paradigma funcional.
    - **Paradigma Funcional: (SUBTIPO DO DECLARATIVO)** estilo de programação com alto nível de abstração, com soluções elegantes, concisas e poderosas. Suas **funcões** computam um resultado que depende apenas dos valores das entradas, ou seja, não existem efeitos colaterais como em programação imperativa. Os programas são **funcões** que descrevem uma relação explícita e precisa entre as entradas e saídas. **Cálculo lambda e Haskell.** Fonte: https://medium.com/@sergiocosta/paradigma-funcional-3194924a8d20
        i. **Visualização uniforme dos programas como funções;**
        ii. **Tratamento das funções como dados;**
        iii. **Limitação do side effect;**
        iv. Uso de gerenciamento de memória automático;
        v. Grande flexibilidade;
        vi. **Notação concisa;**
        vii. Semântica simples.
        viii. Fonte: https://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula15.pdf
4. Explique o uso da Notação lambda (inclua abstração lambda, variáveis livres e associadas).

Sintaxe	Nome	Descrição
x	variável	Caractere que representa parâmetro ou valor
λx. M	abstração	Definição de função sobre a variável x
M N	aplicação	N é aplicado em M
(λx. x+y) z	Tipo de variável	y é livre e x é associada em todo escopo

5. Explica as reduções alfa e beta no contexto do cálculo lambda.

Sintaxe da operação	Nome	Descrição
(λx. M[x]) → (λy. M[y])	α-redução	Renomeia variáveis. Evita conflito de nomes.
((λx.M) E) → (M[x := E])	β-redução	Substitui a variável pelo argumento no corpo da expressão.

6. Explique os tipos de formas funcionais composição, construção e aplica-se a todos.
    - Todas as linguagens funcionais são baseadas no paradigma lambda.
7. Diferencie avaliação ansiosa (eager evaluation) e avaliação tardia/adiada (lazy/delayed evaluation). Fonte: Wikipédia
    - **eager evaluation** (comportamento de avaliação na qual uma expressão é avaliada na primeira vez que é encontrada e seu resultado vinculado a uma variável) is the strategy used by most traditional programming languages. An expression is evaluated as soon as it is bound to a variable. The effects of eager evaluation include:
        i. **Code that is easily understandable in terms of execution order that does not potentially change its behaviour based on a change of execution context.**
        ii. An easier debug process compared to other evaluation strategies due to the above.
        iii. Responsibility for code performance is however shifted towards the programmer, thus requiring a careful code optimisation process.
    - **lazy evaluation** delays the evaluation of an expression until its value is needed (non-strict evaluation) and which also avoids repeated evaluations (sharing). For lengthy operations, it would

be more appropriate to perform before any time-sensitive operations, such as handling user inputs in a video game. The benefits of lazy evaluation include:

   i. The ability to define control flow (structures) as abstractions instead of primitives.
   ii. **The ability to define potentially infinite data structures.** This allows for more straightforward implementation of some algorithms.
   iii. Performance increases by avoiding needless calculations and avoiding error conditions when evaluating compound expressions.

8. Explique o conceito de currying. Quais são suas vantagens? Fonte: Wikipédia
   - **currying** é uma técnica de transformação de uma função que recebe múltiplos parâmetros (mais especificamente, uma n-tupla como parâmetro) de forma que ela pode ser chamada como uma cadeia de funções que recebem somente um parâmetro cada. Foi inventada por Moses Schönfinkel e Gottlob Frege, e independentemente por Haskell Curry.
   - O problema é dividido.
9. Explique a hipótese do mundo fechado no contexto do paradigma lógico.
   - **Paradigma lógico (SUBTIPO DO DECLARATIVO)** faz uso da lógica matemática. **PROLOG.**
   - **Hipótese de mundo fechado:** só se afirma que é verdadeiro o que pode ser provado na base de conhecimentos.
   - O sistema pode supor que conhece todos os fatos verdadeiros a respeito do mundo.
   - O programa é considerado um “mundo fechado” e tudo que a máquina sabe deve estar definido nele.
10. Explique a cláusula Horn e qual é a sua utilidade.

	Implication form	Read intuitively as
Definite clause	$u \leftarrow p \wedge q \wedge ... \wedge t$	u é verdade se p e q .... e t forem verdadeiros
Fact	u	assume u fato verdadeiro

11. Explique o mecanismo de unificação no contexto do paradigma lógico.
    - ??
12. Explique o uso de redes semânticas. Quais são as possíveis cardinalidades dos relacionamentos?
13. Quais são os tempos nos quais podem ocorrer a vinculação (binding)?
    - **Vinculação (binding):** Associação entre entidades de programação ou entre entidades e seus atributos.
        i. Ex.: variáveis com suas propriedades (identificador, tipo, escopo, tempo de vida...).
    - Em tempo de projeto
        i. definição dos símbolos e identificadores que poderão ser usados e das entidades que eles representam.
    - Em tempo de implementação
        i. intervalo de valores associado a um tipo.
        ii. esquemas de representação (diferentes compiladores podem adotar intervalos diferentes).
    - Em tempo de compilação
        i. associação de uma variável a um tipo de dados.
        ii. associação de um símbolo à operação que denota.
    - Em tempo de ligação (linking)
        i. os módulos são integrados para formar um programa executável (ex.: funções em C).
    - Em tempo de carga (load time)
        i. áreas de memória são associadas à variáveis globais ou constantes;

    ii. referências no código viram endereços absolutos.
- Em tempo de execução (run time)
    i. associação de um valor a uma variável;
    ii. associação de áreas de memória com variáveis locais.

14. Diferencie escopo estático e dinâmico, dando exemplos.
    - https://pt.stackoverflow.com/questions/13034/o-que-s%C3%A3o-escopo-l%C3%A9xico-e-escopo-din%C3%A2mico-e-quais-s%C3%A3o-suas-principais-diferen%C3%A7as
    - int x = 0;
    - int f() { return x; }
    - int g() { int x = 1; return f(); } Fonte: Wikipédia
    - Com o **escopo estático, a chamada de g irá retornar 0**, uma vez que foi determinado no momento da compilação que a expressão x em qualquer chamada de f irá produzir uma vinculação global x, que não é afetada pela introdução de uma variável local de mesmo nome em g.
    - Com o **escopo dinâmico,** a pilha de vinculação para o identificador x conterá dois itens quando f é chamada de g: a vinculação global a 0, e a vinculação a 1 introduzida em g (que ainda está presente na pilha uma vez que o fluxo de controle não deixou g ainda). Uma vez que a avaliação da expressão do identificador, por definição, sempre produz a vinculação superior, o resultado **neste caso é 1**.
15. Discuta o tempo de vida de uma variável e seu consequente armazenamento em memória. Apresente vantagens e desvantagens.
    - https://pt.stackoverflow.com/questions/135572/qual-a-diferen%C3%A7a-entre-escopo-e-tempo-de-vida
16. Cite dois exemplos de problemas que podem ocorrer na utilização de ponteiros.
    - Apontar para uma variável que já foi desalocada.
    - Ter dois ponteiros apontando para a mesma variável.