Lista de Exercícios de Linguagens de Programação I
Universidade Federal do Amazonas
Departamento de Ciência da Computação
Marco Cristo

Introdução

1) A principal área de aplicação em meados da década de 50 era a científica, por exemplo, para cálculos balísticos, simulações físicas, etc. Como isso afetou o projeto do Fortran?

   Como aplicações científicas são caracterizadas por tipos numéricos simples envolvidos em operações aritméticas complexas, Fortran foi criada para facilitar a escrita de fórmulas matemáticas e oferecia bom suporte para números reais, vetores e matrizes.

2) Quais as características mais significativas herdadas por Python de Algol?

   Blocos, escopos de ligações, procedimentos & funções, recursão, instruções de iteração.

3) Quais as características mais significativas herdadas por Python de Simula-67?

   Classes.

4) Por que Algol é comumente descrita como um enorme sucesso e, ao mesmo tempo, um enorme fracasso?

   Sucesso por ter sido de enorme influência para outras linguagens (talvez, a linguagem mais influente). Fracasso por ter sido virtualmente não usada.

5) Como a área de aplicação de C (escrita de sistemas operacionais para várias famílias de mini-computadores do início da década de 70) contribuiu para que a linguagem adotasse uma série de características, hoje, consideradas potencialmente inseguras (pouco rigor em checagem de tipos, controle de recursos como memória a cargo do programador, ausência de mecanismos de tratamento de exceções, etc)?

   C precisava ser pequena, eficiente e portável. Vários sacrifícios foram feitos para atingir estas metas.

6) Por que C++, sendo uma linguagem de propósito geral, orientada-para-objetos e desenvolvida no início da década de 90, inclui tantas características de projeto questionáveis de C?

   C++ devia ser completamente compatível com C, para aproveitar a grande base já instalada. Vários sacrifícios foram feitos para atingir esta meta.

7) Qual a diferença entre linguagens de *script* e linguagens de finalidade geral?

   Linguagens de script são destinadas a uma finalidade e contexto particular. Apresentam facilidades que tem sentido naquele contexto, mas que dificultam a sua aplicação eficiente para um contexto diferente do original (mais recentemente, nota-se uma tendência ao surgimento de linguagens de script de propósito cada vez mais geral. Ex: Python, Perl e Ruby).

8) Como você defenderia o uso de interpretação pura para uma linguagem de *script*?

Interpretação pura é de simples implementação e muitos dos contextos específicos em que estas linguagens são usadas não são caracterizados pela necessidade de grande eficiência.

9) Que metas de projeto em Java foram determinantes para que ela fosse projetada como uma linguagem interpretada em uma máquina virtual?

Portabilidade (devia rodar em uma grande variedade de equipamentos) e segurança (deveria ter o mínimo acesso possível a elementos essenciais ao uso seguro/confiável do equipamento).

10) Qual o erro de sintaxe mais comum em programas em LISP?

Casamento de parênteses.

11) A evolução de linguagens de script as têm tornado cada vez maiores e mais complexas: Perl, Python, PHP, etc. Você imagina que está é a tendência para Lua? Suporte a sua resposta:

Provavelmente não. Lua foi projetada para ser pequena de forma a ser utilizada embarcada. Sua idéia é fornecer apenas os elementos essenciais e simples para uma linguagem extensível. Assim, ela possibilitaria a criação de outras linguagens a partir do seu núcleo. Não faz muito sentido adicionar funcionalidades a este tipo de projeto, já que qualquer adição a tornaria maior (indesejável). Também não é provável que teorias significativamente diferente venham a surgir com freqüência que justifiquem mudanças em como se implementa extensibilidade nesta linguagem.

12) Que tipos de novas linguagens surgem mais freqüentemente? De script (LS) ou de propósito geral (LPG)?

LS nascem à medida que surgem novos contextos. LPG nascem na medida em que novos conceitos e paradigmas de programação surgem. Logo LS são mais freqüentes uma vez que há mais novos contextos que novos conceitos e paradigmas.

13) A linguagem C usa "=" para assinalamento e "==" para testar igualdade. Além disso, as duas operações são permitidas em expressões (ex: *a = b = 0*; *a = b == 0*). Os projetistas do C comentam que "desde que assinalamentos são cerca de duas vezes mais comuns que igualdade em programas típicos em C, é apropriado que o operador de assinalamento tenha metade do tamanho" (Kernigan and Ritchie, C the programing language, 1977, pg 17). O que você acha deste argumento para defender a decisão de projeto? Que argumento você usaria para defender ou criticar tal decisão?

Justificativas bizarras não são exclusividade de fanáticos ☺. Não é muito claro se os autores estavam falando sério, realmente. De qualquer modo, esse nível de economia em sintaxe não parece ser de grande valor. Muito mais relevante é a possibilidade de problemas. Note que desde que os operadores são muito parecidos, é fácil que eles sejam trocados um pelo outro, por exemplo, por falha de digitação. E, em C, isso é um problema sério já que qualquer das duas formas é permitida em expressões, o que dificulta muito a detecção do erro.

Compiladores e Gramáticas

14) Defina sintaxe e semântica, em Linguagens de Programação:

Sintaxe = forma → conjunto de regras que definem as combinações de símbolos que são consideradas formas válidas de acordo com a linguagem.
Semântica = significado, correspondendo (a) ao mapeamento de uma sentença de uma linguagem para outra linguagem (denotação), (b) à execução de uma sentença em uma máquina abstrata (operação) e/ou (c) à descrição dos axiomas lógicos que se aplicam a uma sentença (axiomatização).

15) Escreva uma descrição EBNF para a definição de cabeçalho de classe em Java (obs: considere que *Identificador* denota identificadores válidos em Java)
Ex: `public final class MinhaClasse extends MeuPac.Mae implements Interface1, Pacote.Interface2`

```
CabClasse = [Modificadores] “class” Identificador [Super] [Interfaces]
Modificadores = Modificador | Modificadores Modificador
Modificador = “public” | “abstract” | “final”
Super = “extends” TipoClasse
Interfaces = “implements” ListaTiposInterface
ListaTiposInterface = TipoInterface | ListaTiposInterface “,” TipoInterface
TipoClasse = NomeTipo
TipoInterface = NomeTipo
NomeTipo = Identificador | NomePacote “.” Identificador
NomePacote = Identificador | NomePacote “.” Identificador
```

16) Escreva uma descrição EBNF para a definição de cabeçalho de método em Java (obs: considere que *Identificador* denota identificadores válidos em Java e *Tipo* denota os tipos válidos em Java, incluindo nomes válidos de tipos de classe)
Ex: `public static final void func(int n, String results[]) throws ZeroDivisionException, ZClass`

```
CabMetodo = [ModificadoresMetodo] TipoResultado Identificador “(” ListaParametrosFormais “)” [Throws]
TipoResultado = Tipo | “void”
ModificadoresMetodo = ModificadorMetodo | ModificadoresMetodo ModificadorMetodo
ModificadorMetodo = “public” | “protected” | “private” | “static” | “abstract” | “final” | “synchronized” | “native”
Throws = “throws” ListaTiposClasse
ListaParamentrosFormais = ParametroFormal | ListaParamentrosFormais “,” ParametroFormal
ParametroFormal = Tipo Identificador
ListaTiposClasse = TipoClasse “|” ListaTiposClasse “,” TipoClasse
TipoClasse = Tipo
```