Lista de Linguagens de Programação – 7

Nome: _____ Matrícula: _____

1. Dizemos que uma linguagem é segura quando esta linguagem não permite que operações sejam aplicadas a argumentos que não possuam os tipos previstos por estas operações. C e C++ são linguagens inseguras, pois muitas vezes valores armazenados em memória são utilizados sem qualquer fiscalização de seus tipos.

   (a) Escreva um programa em C ou C++ que evidencie o caráter inseguro de uma dessas linguagens.

   (b) Existem linguagens mais antigas que C ou C++ que são consideradas seguras, logo, a possibilidade de uso inseguro de tipos não é devido à ignorância sobre os perigos dessa abordagem. ML, por exemplo, já havia sido definida dez anos antes de C++, porém enquanto ML é uma linguagem considerada segura, C++ não é. Cite um fator que motivou o desenho inseguro de C++.

2. Essa questão refere-se ao programa abaixo, escrito na linguagem C. Esse programa foi compilado com `gcc 4.2`, e testado em um sistema operacional Mac OS X 10.5.8:

```
1  include <stdio.h>
2  void function() {
3    int buffer[1];
4    buffer[4] += 3;
5  }
6  int main() { int x; x = 13; function(); x++; printf("%d\n",x); }
```

   (a) Compile e execute esse programa. O que será impresso?

   (b) Comente a linha 4 do código fonte de nosso programa, compile-o novamente e re-execute. O que será impresso?

   (c) Compile esse programa com informações de depuração usando o comando `gcc -g f.c`, e o execute com o depurador gdb, via o comando `gdb a.out`. No terminal do depurador, execute os seguintes comandos:

```
(gdb) b function
(gdb) run
(gdb) print buffer[4]
$1 = 8156              // <--- provavelmente voce obterah outro valor
(gdb) x 8152           //      aqui. Adapte sua saida de acordo.
(gdb) disassemble main
```

   O que é o valor armazenado em `buffer[4]`?

   (d) Qual o efeito da atribuição `buffer[4] += 3`?

   (e) Esse exercício ilustra uma vulnerabilidade de programas escritos na linguagem C, chamada *buffer overrun*. Por que essa vulnerabilidade não ocorre em linguagens fortemente tipadas?

3. Uma linguagem é estaticamente tipada quando o tipo de cada expressão pode ser resolvido em tempo de compilação. Uma linguagem é dinamicamente tipada quando o tipo da variável é resolvido em tempo de execução. Neste exercício reforçaremos estes conceitos.

   (a) Dê um exemplo de uma linguagem estaticamente tipada. Como o compilador consegue descobrir o tipo das variáveis, no caso desta linguagem?

   (b) Dê um exemplo de uma linguagem dinamicamente tipada. Escreva um programa, muito simples, que evidencie o caráter dinâmico desta linguagem.

   (c) Cite uma vantagem da tipagem estática sobre a tipagem dinâmica.

   (d) Agora, cite uma vantagem da tipagem dinâmica sobre a tipagem estática.

4. O objetivo deste exercício é completar a função abaixo, que computa o `Crivo de Erastótenes`:

```
fun filterNonPrimes _ nil = 0
  | filterNonPrimes limit (h::t) =
      if h * h <= limit
      then h + filterNonPrimes limit (filter (fn e => (e mod h) <> 0) t)
      else h + sum t

fun sieve n =  filterNonPrimes n (inv (range n) nil)
```

   (a) Escreva a função `sum`, de tipo `int list -> int`, que calcula a soma de uma lista de inteiros.

   (b) Escreva a função `range`, de tipo `int -> int list`, que produza listas de inteiros em ordem descrescente, isto é, `range 4 = [4,3,2]`.

   (c) Escreva a função `inv`, de tipo `'a list -> 'a list -> 'a list`, que receba duas listas: l_1 e l_2. A função deve inverter a lista l_1, usando a lista l_2 como um acumulador da lista invertida. Isto é, `inv [4,3,2] nil = [2,3,4]` e `inv [5, 4, 3] [8, 9, 10] = [3, 4, 5, 8, 9, 10]`. Note que o propósito do parâmetro l_2 é tornar a implementação da função mais eficiente.

5. O autor de ML, *Robin Milner*, ganhou um prêmio *Turing*. Uma de suas maiores contribuições à ciência da computação foi um algoritmo para a *inferência de tipos*. ML é uma linguagem estaticamente tipada, porém o desenvolvedor em geral não precisa escrever o tipo durante a declaração de expressões. Os tipos são inferidos automaticamente pelo compilador ou pelo interpretador. O objetivo deste exercício é fazer com que você pense como um compilador, quando este realiza a inferência de tipos. Tente descobrir o tipo de cada uma das funções abaixo. Assim que achar que tem uma resposta, escreva a função em SML, e verifique se a resposta está correta. Escreva o tipo de cada função.

   (a) `f(x) = 1`

   (b) `f(x, y) = 1`

   (c) `f(x) = x`

   (d) `f(x, y) = x`

   (e) `f(g) = g(1)`

   (f) `f(g, x) = g(x)`

   (g) `f(g, x, y) = g(x, y)`

   (h) `f(g, h, x) = g(h(x))`

   (i) `f(g, x) = g(g(x))`

6. Essa questão refere-se à seção de Python abaixo, em que se define e usa-se uma função fatorial.

```
~/fernando$ python
Python 2.5.1 (r251:54863, May  5 2011, 18:37:34)
>>> def fact(n):
...   if n > 1:
...     return n * fact(n-1)
...   else:
...     return 1
...
>>> type(fact(10))
<type 'int'>
>>> type(fact(100))
<type 'long'>
>>>
```

(a) Quando o tipo de `fact(10)` é conhecido? As duas opções são: (i) em tempo de compilação da função `fact`, quando sua declaração passa a ser conhecida, ou, (ii) durante a execução do programa, quando a chamada `fact(10)` é feita.

(b) Por que o tipo de `fact(10)` é diferente do tipo de `fact(100)`?

(c) Considere a função abaixo, escrita na linguagem C. Qual é, nesse caso, o tipo das variáveis `x` e `y`?

```
int fact(int n) { return n > 1 ? n * fact(n-1) : 1; }
void main() {
  int x = fact(10);
  int y = fact(100);
  printf("%d, %d\n", x, y);
}
```

(d) Do ponto de vista de facilidade de programação, qual forma de tipagem torna o programador mais produtivo?