Primeira Prova de Linguagens de Programação
- DCC024 -
Ciência da Computação

Nome: _____
"Eu dou minha palavra de honra que não trapacearei neste exame."

Número de matrícula: _____

As regras do jogo:

- A prova é sem consulta.
- Quando terminar, não entregue nada além do caderno de provas para o instrutor.
- Quando escrever código, a sintaxe correta é importante.
- Cada estudante tem direito a fazer uma pergunta ao instrutor durante a prova. Traga o caderno de provas quando vier à mesa do instrutor.
- A prova termina uma hora e quarenta minutos após seu início.

Alguns conselhos:

- Escreva sempre algo nas questões, a fim de ganhar algum crédito parcial.
- Se não entender a questão, e já tiver gasto sua pergunta, escreva a sua interpretação da questão junto à resposta.
- Serão avaliadas somente as sete melhores respostas. Então sinta-se livre para abandonar alguma questão devido ao tempo.
- A prova não é difícil, ela é divertida, então aproveite!

Tabela 1: Pontos acumulados

Questão 1	Questão 2	Questão 3	Questão 4	Questão 5	Questão 6	Questão 7	Questão 8

1. Esta questão refere-se à gramática abaixo, que representa uma linguagem muito simples, de somas de números. Por simplicidade nós não mostraremos as regras de produção para números:

$$\begin{array}{rcl} \langle E \rangle & ::= & \langle E \rangle + \langle E \rangle \\ & | & \langle Number \rangle \end{array}$$

   (a) Prove que a gramática em questão é ambígua.

   (b) Mostre como esta ambiguidade compromete a semântica da linguagem que a gramática representa.

   (c) Forneça uma gramática que reconheça a mesma linguagem, mas que não seja ambígua.

2. O programa `gcc`, usado em Unix para invocar o compilador de C desenvolvido pela gnu, é na verdade um *script* que invoca vários outros programas. É possível saber quais os programas são invocados por `gcc` adicionando-se o parâmetro `-v` à sua linha de invocação. Cada uma das questões abaixo contém um dos passos adotados por `gcc` para produzir um arquivo binário a partir de um programa fonte `hello.c`. Descreva o que cada uma destas linhas faz.

   (a) `gcc -E hello.c > hello.p.c`

   (b) `gcc -S hello.p.c -o hello.p.s`

   (c) `as hello.p.s -o hello.o`

   (d) `/usr/bin/ld hello.o -o a.out`

3. 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 destas 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 desta 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++.

4. Nós podemos representar números inteiros usando o cálculo λ. Uma das convenções mais comuns é assumir que um número n é uma função que recebe dois argumentos, e aplica o primeiro ao segundo n vezes. Por exemplo:

   - $0 = \lambda s.\lambda z.z$
   - $1 = \lambda s.\lambda z.sz$
   - $2 = \lambda s.\lambda z.s(sz)$

   Podemos também representar valores booleanos usando o cálculo λ. Uma convenção simples é:

   - $F = \lambda x.\lambda y.y$
   - $T = \lambda x.\lambda y.x$

   (a) Considere a função MUL = $\lambda n_1.\lambda n_2.\lambda z.n_1(n_2\ z)$. Usando a definição do número 2 acima, mostre todos os passos da redução MUL 2 2.

   (b) Usando a função sucessor, SUCC = $\lambda n.\lambda y.\lambda x.y(n\ y\ x)$, defina a função ADD, que soma dois números.

   (c) Defina uma função Z, que receba um argumento n. Assuma que este argumento é um número, representado segundo a convenção acima. A função Z deve retornar o booleano T caso este número seja zero, ou seja, $n = \lambda s.\lambda z.z$, e deve retornar F caso contrário.

   (d) Defina uma função XOR, que receba dois valores booleanos b_1 e b_2, definidos como convencionado acima, e retorne T caso $b_1 \neq b_2$ e F caso contrário.

5. Considere o programa abaixo, escrito em SML/NJ:

```
fun g x =
  let
    val inc = 1
    fun f y = y + inc
    fun h z =
      let
        val inc = 2
      in
        f z
      end
  in
    h x
  end
```

(a) Desenhe um círculo em torno de cada bloco deste programa, e numere estes blocos.

(b) Quais são os nomes definidos neste programa?

(c) Para cada definição, descreva o escopo desta definição fornecendo seu número de bloco.

(d) Para cada ocorrência de um nome, além da definição daquele nome, mostre a qual definição este nome está associado.

(e) Com base no resultado da questão anterior, qual o valor de `g 5`?

6. O objetivo desta questão é escrever uma função `concat` de várias formas diferentes. Esta função recebe como entrada uma lista de strings, e retorna uma única string, formada a partir da concatenação dos elementos da lista. Por exemplo, `concat ["ab", "cd", "ef"] = "abcdef"`.

   (a) Escreva a função `concat` usando recursão explícita. Neste caso, programe "indutivamente", isto é, defina um caso base, quando a lista de entrada estiver vazia, e defina um caso de indução. No passo indutivo, use o seguinte raciocínio para escrever o programa: dado que você sabe concatenar uma lista de n elementos, como fazer para concatenar uma lista de $n + 1$ elementos?

   (b) Escreva a função `concat` em uma linha, usando a função `foldr`.

7. As funções deste exercício devem ser escritas sem que sejam utilizadas as funções `foldr`, `foldl` e `map`.

   (a) Defina a função `mymap`, que tenha o mesmo tipo e comportamento de `map`.

   (b) Qual o tipo de `mymap`?

   (c) Defina a função `myfoldr`, que tenha o mesmo tipo e comportamento de `foldr`.

   (d) Qual o tipo de `myfoldr`?

   (e) Defina a função `myfoldl`, que tenha o mesmo tipo e comportamento de `foldl`.

8. 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.