Primeira Prova de Linguagens de Programação
- DCC024 -
Sistemas de Informação

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 as seis 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

1. Esta questão refere-se à gramática abaixo, escrita em Prolog:

```
expr --> mulexp, [+], expr.
expr --> mulexp.
mulexp --> rootexp, [*], mulexp.
mulexp --> rootexp.
rootexp --> ['('], expr, [')'].
rootexp --> number.
number --> digit.
number --> digit, number.
digit --> [0] ; [1] ; [2] ; [3] ; [4] ; [5] ; [6] ; [7] ; [8] ; [9].
```

   A título de exemplo, a expressão $12 \times (3 + 56)$ seria escrita, nesta gramática, assim:
   `expr([1,2,*,'(',3,+,5,6,')'], []).`

   (a) (4 pontos) Modifique a gramática para que ela passe a tratar de números com sinal. Por exemplo, a expressão $-12 \times (3 + -56)$ seria escrita assim: `expr([-,1,2,*,'(',3,+,-,5,6,')'], [])`. Não é necessário re-escrever toda a gramática. Re-escreva somente as partes que serão modificadas.

   (b) (3 pontos) Adicione à gramática original (não à gramática da última questão) um operador binário `&`, que seja associativo à esquerda, e que tenha precedência menor que todos os outros operadores da gramática. Por exemplo, a expressão $12 \times (3\&56)$ seria representada como `expr([1,2,*,'(',3,&,5,6,')'], [])`..

   (c) (3 pontos) Adicione à gramática original um operador unário **, que seja associativo à direita, e que tenha a precedência maior que todos os outros operadores, menos os parênteses. Por exemplo, a expressão $12 \times 3^2 + 56$ seria representada como `expr([1,2,*,'(',**,3,+,5,6,')'], [])`..

2. (2 pontos cada) 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 programador. Neste exercício você deverá pensar como um compilador, quando este realiza a inferência de tipos. Escreva o tipo de cada função abaixo.

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

   (b) `f(x) = x`

   (c) `f(g) = g(1)`

   (d) `foldr (fn(x,y) => x + y)`

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

3. (2.5 pontos cada questão) Linguagens de programação utilizam vários mecanismos para descobrir os tipos dos valores manipulados nos programas. Abaixo temos três implementações do programa fatorial, escritas em três linguagens diferentes. Para cada implementação, responda às seguintes perguntas:

   - Quando o tipo do nome n é conhecido? Durante a compilação do programa, ou durante a sua execução?
   - Como a implementação da linguagem descobre o tipo de n?

   (a) O programa abaixo foi escrito em C:

```
#include <stdio.h>
int fact(int n) {
  int f = 1;
  while (--n)
    f *= n;
  return f;
}
int main(int argc, char** argv) {
  printf("\%d\n", fact(argc));
}
```

   (b) O programa abaixo foi escrito em SML:

```
fun fact n = if n > 1 then n * fact(n-1) else 1;
```

   (c) O programa abaixo foi escrito em javaScript:

```
function fact(n) {
  if (n > 1) {
    return n * fact(n - 1);
  } else {
    return 1;
  }
}
print(fact("Love is the only reason to live."));
print(fact(10));
```

   (d) O programa acima, em javaScript, produz a seguinte saída:

```
1
3628800
```

   Com base nesta informação, javaScript é uma linguagem fortemente ou fracamente tipada?

4. (2.5 pontos cada questão) Esta questão refere-se às seguintes definições de funções λ:

   - Pair = $\lambda a\ .\ \lambda b\ .\ \lambda f\ .\ fab$
   - Head = $\lambda g\ .\ g(\lambda a\ .\ \lambda b\ .\ a)$
   - Tail = $\lambda g\ .\ g(\lambda a\ .\ \lambda b\ .\ b)$
   - Curry = $\lambda f\ .\ \lambda x\ .\ \lambda y\ .\ f(\text{Pair}\ x\ y)$
   - Uncurry = $\lambda f\ .\ \lambda p\ .f(\text{Head}\ p)(\text{Tail}\ p)$

   Com base nas definições acima, mostre todos os passos das reduções abaixo:

   (a) Tail(Pair p q) $\Rightarrow q$

   (b) Curry (Uncurry h) $\Rightarrow h$

   (c) Uncurry (Curry h) (Pair r s) $\Rightarrow h$(Pair r s)

   (d) Como representar listas no cálculo lambda, usando as definições de Pair, Head e Tail? Nesta questão, apenas apresente sua idéia. Não é necessário mostrar detalhadas equações lambda. Para ilustrar sua abordagem, descreva como a lista [1,2,3,4] seria representada.

5. Linguagens de programação provêem aos desenvolvedores diversas abstrações que têm o propósito específico de permitir reúso de código. As questões abaixo referem-se a duas destas abstrações:

   (a) Escreva um programa em SML que ilustre como funções de alta ordem são utilizadas como um mecanismo de reúso. Explique o que está sendo reusado neste exemplo.

   (b) Escreva um programa em SML que mostre como o polimorfismo paramétrico possibilita o reúso de código. Assim como na questão anterior, descreva o que está sendo reusado.

6. A solução esperada para os três próximos exercícios deve ter somente uma linha, e deve usar as funções `map`, `foldr` e `foldl`.

   (a) (4 pontos) Escreva uma função `squareList`, de tipo `int list -> int list` que receba uma lista de inteiros e retorne uma lista dos quadrados destes inteiros. Por exemplo, a chamada `squareList [1, 2, 3, 4]` deveria retornar `[1, 4, 9, 16]`.

   (b) (3 pontos) Escreva uma função `sqSum`, de tipo `int list -> int` que receba uma lista de inteiros e retorne a soma dos quadrados destes inteiros. Por exemplo, a chamada `sqSum [1, 2, 3, 4]` deveria retornar `30`.

   (c) (3 pontos) Escreva uma função `dupList`, de tipo `'a list -> 'a list`, cujo resultado seja a lista de entrada com cada elemento repetido em sequência. Por exemplo, a chamada `dupList [1, 2 ,3]` deveria retornar `[1, 1, 2, 2, 3, 3]`. Se a lista de entrada é nula, o resultado deve ser a lista nula.

7. As duas questões a seguir referem-se ao registro de ativação de funções. Registros de ativação são os blocos de dados utilizados para armazenar as informações necessárias à execução das funções. Em geral estes blocos são criados durante a chamada da função, e colocados em uma pilha. Quando a função termina, seu registro de ativação é desempilhado. A alocação, neste caso, tende a ser bastante eficiente, pois a última função chamada será a primeira a terminar.

   (a) (2 pontos) Que informações são normalmente mantidas em registros de ativação de funções implementadas na linguagem ANSI C?

   (b) (3 pontos) Cite uma diferença entre os registros de ativação usados na linguagem ANSI C e em SML.

   (c) (5 pontos) Considere a função `foo`, abaixo, cujo tipo é `int -> int list -> int list`.

```
fun foo x nil = nil
  | foo x (h::t) =
    let
      fun addX y = y + x
    in
      addX h :: foo x t
    end
```

   Esta função recebe um número inteiro `x` e uma lista e soma este número a cada elemento da lista, por exemplo, `foo 1 [1,2] = [2,3]`. Desenhe os vários registros de ativação criados devido à chamada `foo [1, 2]`. Estes registros devem ser desenhados imediatamente antes que a última invocação da função `addX` termine. Isto é, desenhe os registros de ativação no momento em que o registro criado para a chamada `addX 2` está a ponto de ser desempilhado.