Primeira Prova de Linguagens de Programação
- DCC024B -
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.
- Seja honesto e lembre-se: **você deu sua palavra de honra**.

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.
- A prova não é difícil, ela é divertida, então aproveite!

Tabela 1: Pontos acumulados (para uso do instrutor)

Questão 1	Questão 2	Questão 3	Questão 4	Questão 5

Questão Extra (0.5 Pontos): **Escolha somente um desafio**: (i) Em que país nasceu o cientista que é creditado como o inventor do algoritmo *merge sort*? (ii) Cite seis estórias de *Hans Christian Andersen*.

1. Esta questão diz respeito a uma função `oneTrue`, que, quando implementada em SML, recebe um predicado e uma lista, e retorna verdade se existe, na lista, algum elemento que torna verdadeiro o predicado. O tipo dessa função é `('a -> bool) -> 'a list -> bool`. Por exemplo:

```
- oneTrue (fn x => x < 3) [4, 5, 6];
val it = false : bool

- oneTrue (fn x => x < 3) [1, 5, 2, 6];
val it = true : bool

- oneTrue (fn x => x = "oi") ["ai", "ei", "oi"];
val it = true : bool
```

   (a) (5 Pontos) Implemente essa função em SML, sem usar as funções de alta ordem `map`, `foldr` ou `foldl`. Não assuma a existência de qualquer função pré-definida em SML.

   (b) (5 Pontos) Implemente `oneTrue` em uma linha de código, usando para tanto qualquer combinação das funções `map`, `foldr` e `foldl`. Você não precisa usar todas essas funções, mas deve usar pelo menos uma delas.

2. Esta questão compara dois programas de semântica similar, escritos em C e Java. Para responder as perguntas que se seguem, procure ter em mente conceitos como *registro de ativação* e *Tipagem Fraca/Forte*.

   (a) (5 Pontos) Considere o programa abaixo:

```
#include <stdio.h>
void function() {
  int v[0];
  v[3] = v[3] + 4;
}
int main() {
  int x;
  x = 13;
  function();
  x++;
  printf("%d\n",x);
}
```

   Esperar-se-ia que o programa acima imprimisse a saída `14`. Porém, ao ser executado em uma máquina Intel x86 com sistema operacional Linux Ubuntu, o programa imprime `13` e termina normalmente. Porque o programa imprime `13`, e não `14`, como seria esperado?

   (b) (5 Pontos) Considere agora, o programa abaixo, escrito em Java. Seria possível obter a mesma saída que aquela produzida pelo programa em C acima, desta vez usando o programa escrito em Java?

```
public class Buf {
  public static void function() {
    int[] v = new int[0];
    v[3] = v[3] + 4;
  }
  public static void main(String args[]) {
    int x;
    x = 13;
    function();
    x++;
    System.out.println("" + x);
  }
}
```

3. Esta questão faz referência a diferentes representações de valores no cálculo λ.

   (a) (5 Pontos) Podemos representar números em cálculo λ como funções que recebem "dois parâmetros", s e z. A representação do número n aplica a função s sobre o parâmetro z um número de vezes igual a n. Abaixo vemos algumas representações de números:

$$\begin{aligned} 0 &= \lambda s.\lambda z.z \\ 1 &= \lambda s.\lambda z.sz \\ 2 &= \lambda s.\lambda z.s(sz) \\ 3 &= \lambda s.\lambda z.s(s(sz)) \end{aligned}$$

   Tais números são chamados *Números de Church*, em homenagem a Alonzo Church, o cientista que inventou o cálculo λ. É possível gerar qualquer número via essa representação, usando-se para isso a função $SUCC$, que recebe a representação do número n, e produz a representação do número $n + 1$:

$$SUCC = \lambda n.\lambda y.\lambda x.y(nyx)$$

   Dada essa representação de número, escreva a função ADD, que receba dois números de Church, n_1 e n_2, e produza a representação de $n_1 + n_2$. Comece sua função assim: $ADD = \lambda n_1.\lambda n_2.\ldots$.

   (b) (5 Pontos) É possível criar diferentes representações para os valores booleanos. Uma representação bastante adotada é a seguinte:

$$\begin{aligned} True &= \lambda x.\lambda y.x \\ False &= \lambda x.\lambda y.y \end{aligned}$$

   Dada essa representação de valores booleanos, escreva uma função AND, que receba dois booleanos, b_1 e b_2, e produza a conjunção lógica desses booleanos, segundo a tabela:

$$\begin{array}{llll} AND & True & True & = True \\ AND & True & False & = False \\ AND & False & True & = False \\ AND & False & False & = False \end{array}$$

   Comece a sua função com a declaração: $AND = \lambda b_1.\lambda b_2.\ldots$.

4. Esta questão refere-se ao tipo algébrico abaixo, que representa árvores de dois ou mais nodos:

```
datatype 'a Tree = Leaf
                 | BinNode of 'a Tree * 'a * 'a Tree
                 | ListNode of 'a * 'a Tree list ;
```

Vemos, abaixo, a definição de algumas instâncias do tipo `Tree`:

```
- val T0 = BinNode (Leaf, "oi", Leaf);
- val T1 = BinNode (Leaf, "ei", T0);
- val T2 = ListNode ("ui", [T0, T1]);
- val T3 = ListNode ("ai", [Leaf, T0, T1, T2, T0]);
```

(a) (2 Pontos) A definição de `Tree` usa um tipo de polimorfismo muito comum em SML. Cite uma outra linguagem de programação que possui essa forma de polimorfismo. Para que sua questão seja avaliada, você precisa mostrar um exemplo de uso desse polimorfismo na linguagem escolhida.

(b) (5 Pontos) Escreva uma função `contains`, de tipo `''a -> ''a Tree -> bool`, que receba um elemento e uma árvore, e retorne verdadeiro caso o elemento esteja na árvore. A sua função deve retornar falso caso o elemento não esteja presente na árvore. Por exemplo, considerando-se as definições acima, teríamos:

```
- contains "ai" T3 ;
val it = true : bool
- contains "ui" T3;
val it = true : bool
- contains "ui" T1;
val it = false : bool
```

(c) (3 Pontos) o tipo de contains é `''a -> ''a Tree -> bool`. O que significa o duplo apóstrofo na definição de `a`?

5. Esta questão refere-se à gramática abaixo, que descreve expressões aritméticas envolvendo somas e multiplicações de números binários:

```
expr --> mulexpr, [+], expr.
expr --> mulexpr.
mulexpr --> number, [*], mulexpr.
mulexpr --> number.
number --> digit, number.
number --> digit.
digit --> [0].
digit --> [1].
```

(a) (3 Pontos) Desenhe todas as árvores de derivação possíveis que a gramática acima pode gerar para a cadeia de caracteres: `1, +, 1, +, 1`. Informe explicitamente, quantas árvores você encontrou.

(b) (2 Pontos) Qual dos operadores, (`+`) ou (`*`) possui maior precedência? Justifique a sua resposta.

(c) (5 Pontos) Modifique a gramática acima, para que ela seja capaz de computar o valor de expressões aritméticas na base 2. Efetivamente, você deverá adicionar atributos à gramática. Esses atributos irão computar o valor de cada cadeia válida de caracteres. Por exemplo:

```
?- expr(N, [1, 1, 0, *, 1, 0], []).
N = 12 ;
false.

?- expr(N, [1, 1, 0, +, 1, 0], []).
N = 8 ;
false.
```

No exemplo acima, o atributo `N` guarda o valor, em base 10, representado pelas *strings* escritas em base 2. Em outras palavras, $110_2 + 10_2 = 1100_2 = 12_{10}$.

6. O *Link de Aninhamento* é uma estrutura usada para permitir que uma função aninhada possa fazer referência a variáveis declaradas no corpo da função aninhadora. Como um exemplo, considere o programa abaixo, implementado em **Gnu C**:

```
int sum(int x, int limit) {
  int prod(int begin) {
    if (begin >= limit)
      return 1;
    else
      return begin * prod(begin + 1);
  }
  return prod(x);
}
int main(int argc, char** argv) {
  printf("sum(2, 1) = %d\n", sum(2, 1));
}
```

A função `prod` precisa utilizar um link de aninhamento para encontrar o valor da variável `limit`, pois essa variável não é local a `prod`.

(a) Em cada uma das situações abaixo, determine como é descoberto o endereço para onde aponta o link de aninhamento da função g, logo que ela é ativada:

i. (2 Pontos) A função g não está aninhada dentro de alguma outra função.

ii. (2 Pontos) g está aninhada dentro de uma função f, e g é chamada pela primeira vez.

iii. (2 Pontos) A função g está aninhada dentro de uma função f, e g é chamada recursivamente.

(b) (4 Pontos) O link de aninhamento é necessário em linguagens de programação que permitem a declaração de funções aninhadas. Por outro lado, mesmo linguagens que não possuem esse recurso, como **ANSI C**, ainda permitem que variáveis livres existam. Isso ocorre devido às variáveis globais:

```
int counter;
void incCounter() {
  counter++; // counter eh variavel livre no escopo de incCounter
}
int main() {
  counter = 0;
  incCounter();
  printf("Counter = %d\n", counter);
}
```

Por que não é necessário um link de aninhamento para encontrar-se o valor de variáveis globais?