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 6

1. (10 Pontos) Esta questão concerne a gramática abaixo, que é *ambígua*:

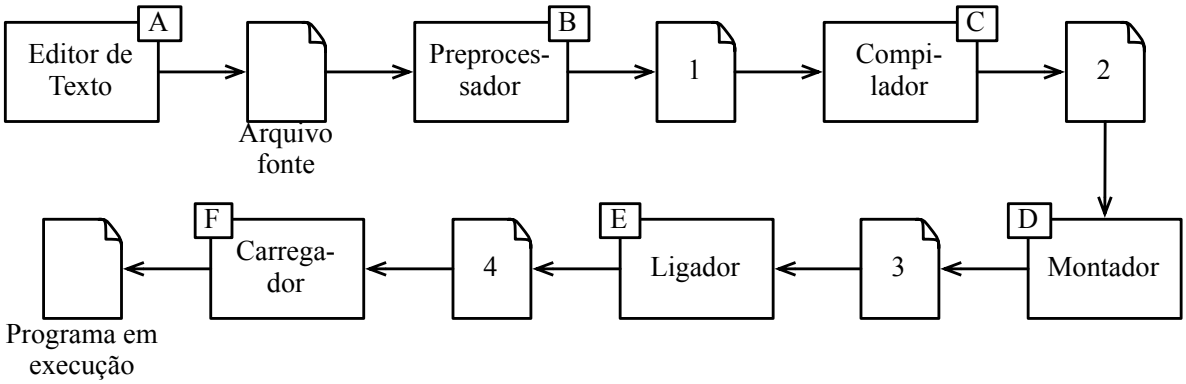
```
cmd --> [if], bool_expr, [then], cmd.
cmd --> [if], bool_expr, [then], cmd, [else], cmd.
cmd --> [halt].
```

   (a) Prove que a gramática acima é ambígua.

   (b) Como essa ambiguidade compromete a semântica dos programas?

   (c) SML possui um construto do tipo *if-then-else*, porém a gramática de SML não é ambígua. Por que essa ambiguidade não existe em SML?

2. (10 Pontos) Essa questão refere-se ao diagrama abaixo, que mostra o ciclo de vida de um programa escrito na linguagem C:



- (4 pontos) O arquivo fonte é uma sequência de *caracteres ASCII* que formam um programa correto na *linguagem C*. Para cada uma das outras representações intermediárias que o programa assume, diga se o formato é ASCII ou binário, e a linguagem (*C, de montador, de máquina*) em que tal arquivo é escrito.
  - **1:**
  - **2:**
  - **3:**
  - **4:**
- (2 pontos) Caso mudemos a arquitetura alvo do diagrama acima, por exemplo, de x86 para PowerPC, quais representações de programa certamente mudariam?

- (2 pontos) A maior parte dos formatos acima seguem gramáticas idênticas, ou muito parecidas. Existe, contudo, uma das fases do processo de compilação, *A, B, C, D, E* ou *F*, que muda a gramática da representação intermediária consideravelmente. Qual fase é essa?

- (2 pontos) Caso mudássemos a linguagem, de C, para Java, qual seria o formato e linguage de entrada e o formato e linguagem de saída da fase C do diagrama acima?

3. Considere a assinatura abaixo, que descreve a interface de um conjunto de inteiros:

```
signature SET =
sig
  type set
  val new : set
  val add : set -> int -> set
  val contains : set -> int -> bool
  val remove : set -> int -> set
  val union : set -> set -> set
  val intersection : set -> set -> set
  val complement : set -> set
end ;
```

Uma possível implementação para essa interface é dada logo abaixo. Nesse caso, estamos usando funções para representar os conjuntos.

```
structure FunSet :> SET =
struct
  type set = int -> bool
  val new = fn x => false
  fun add f i = fn x => if x = i then true else f x
  fun contains f i = f i
  fun remove f i = fn x => if x = i then false else f x
  fun union ...
  fun intersection ...
  fun complement ...
end ;
```

- (3 pontos) Complete a função `union`, de tipo `set -> set -> set` na estrutura `FunSet`. `FunSet.union f1 f2` retorna um novo conjunto que contém qualquer elemento que esteja em `f1` ou em `f2`.

- (3 pontos) Complete a função `intersection`, de tipo `set -> set -> set` na estrutura `FunSet`. `FunSet.intersection f1 f2` retorna um novo conjunto que contém somente os elementos que estiverem tanto em `f1` quanto em `f2`.

- (4 pontos) Complete a função `complement`, de tipo `set -> set` na estrutura `FunSet`. `FunSet.complement f` retorna um novo conjunto `f'` que contenha (dentro do universo dos inteiros de SML) cada elemento que `f` não contiver, e vice-versa.

4. (10 pontos) Podemos representar os valores booleanos no cálculo lambda segundo a seguinte convenção:

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

   Dessa forma, o valor verdade (T) é uma função que recebe dois parâmetros [1] x e y, e retorna o primeiro. Já o valor falso é uma função, que, tal qual T, também recebe dois parâmetros, porém retorna o segundo deles. Nesta questão, pede-se que seja implementada – em cálculo lambda – a função lógica `NAND`, tal que:

   - $nand \ T \ T = F$
   - $nand \ T \ F = T$
   - $nand \ F \ T = T$
   - $nand \ F \ F = T$

   Não assuma a existência de nenhuma função pronta, mas sinta-se livre para implementar, sempre em cálculo lambda, quaisquer funções auxiliares que forem necessárias.

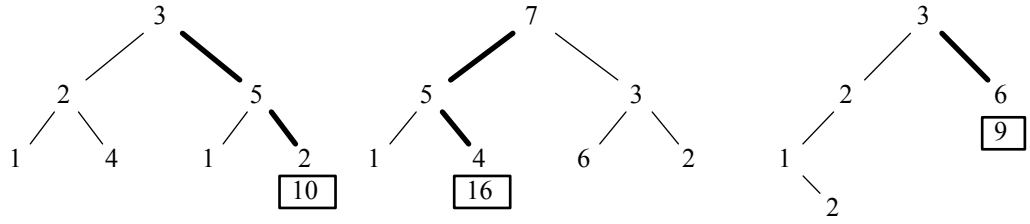
[1] O "recebe dois parâmetros"é, na verdade, um abuso de linguagem que descreve uma função de alta ordem, pois toda função no cálculo lambda recebe somente um parâmetro.

5. (10 pontos) Escreva o mais curto exemplo, em SML/NJ, em que você consiga pensar, que não funcionaria corretamente caso SML/NJ não implementasse registros de ativação usando links de aninhamento. Isto é, os registros de ativação podem ser implementados via uma pilha, como é normalmente feito, mas não possuem link de aninhamento.

6. (10 pontos) Considere o tipo algébrico `tree`, que está definido abaixo:

```
datatype 'data tree =
   Empty |
   Node of 'data tree * 'data * 'data tree
```

Escreva uma função `bPath`, de tipo `int tree -> int`, que retorne a soma do caminho de maior soma na árvore de entrada. Por exemplo, abaixo temos algumas árvores, com seu caminho mais pesado marcado, e a soma desse caminho dada.



Por exemplo:

```
- val T = Node(Empty, 4, Node(Empty, 3, Empty));
val T = Node (Empty,4,Node (Empty,3,Empty)) : int tree

- bPath T;
val it = 7 : int
```