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	Extra

Questão Extra (0.5 Pontos): cite um livro que já recebeu o Prêmio Pulitzer, em qualquer categoria.

1. Assim como os arranjos usados em linguagens imperativas (por exemplo: C, C++ e Java) as tuplas presentes em ML também permitem acesso aleatório em tempo constante. Esta questão faz um contraponto entre arranjos em C e tuplas em ML.

   (a) (3 Pontos) Complete a definição da função `inc_n`, escrita na linguagem C. Esta função recebe um arranjo de inteiros `v`, mais um inteiro `n`, e incrementa o n-ésimo elemento de `v`. A função retorna o arranjo atualizado.

   ```
   int* inc_n(int* v, int n) {



   }
   ```

   (b) (7 Pontos) É possível escrever uma função em ML que incrementa o n-ésimo elemento de uma tupla contendo k inteiros, $k > n$? **Em caso afirmativo**, escreva uma função `inc_n` que receba uma tupla t, contendo k elementos, mais um inteiro n. A função `inc_n` deve retornar uma nova tupla t', que seja igual a t, exceto quanto ao índice n. Naquele índice, tem-se que: $\#n\ t' = 1\ +\ \#n\ t$. **Em caso negativo**, explique porque a sua função não pode ser construída em ML.

2. Segundo o *Princípio da Substituição de Liskov*, um tipo S é subtipo de um tipo T, caso elementos do tipo S possam ser usados em qualquer situação em que elementos do tipo T são esperados. Por exemplo, abaixo temos um programa escrito em `Scala`, que exercita esse princípio:

```
scala> class Animal { val sound = "grrr" }
scala> class Bird extends Animal { override val sound = "piu-piu" }
scala> class Chicken extends Bird { override val sound = "poh-poh-poh" }
scala> def getSound(a: Animal) = a.sound
scala> val a0 = new Animal
scala> val b0 = new Bird
scala> val c0 = new Chicken
scala> getSound(a0)
$> grrr
scala> getSound(b0)
$> piu-piu
scala> getSound(c0)
$> poh-poh-poh
```

   (a) (2 Pontos) Porque o programa acima, escrito em `Scala`, exercita o princípio da Substituição de Liskov?

   (b) (8 Pontos) O Princípio da Substituição também aplica-se a funções. Seja `f : T0 => T1` uma função que receba um argumento do tipo `T0`, e retorne um argumento do tipo `T1`, e seja `f': T0' => T1'` uma função que receba um argumento do tipo `T0'`, e retorne um argumento do tipo `T1'`. Quais as relações de subtipagem que devem existir entre `T0`, `T1`, `T0'` e `T1'`, para que a função `f'` seja subtipo da função `f`? Para lhe ajudar a pensar na resposta, considere o seguinte programa, também em `Scala`:

```
scala> def getBirdSound(b: Bird) = b.sound
scala> def doSomething(f: Bird => String) = {val c = new Chicken; println(f(c))}
scala> doSomething(getBirdSound)
$> poh-poh-poh
```

   Procure pensar em quais funções poderiam ser passadas para `doSomething`. Mas, sua resposta ainda precisa ser genérica o suficiente: responda a questão em termos de `T0`, `T1`, `T0'` e `T1'`.

3. Nesta questão você deverá escrever uma função `middle`, cujo tipo é `'a list -> 'a`. Essa função recebe uma lista `L` e retorna o elemento no meio de `L`. Por exemplo:

```
- middle ["a", "e", "i", "o", "u"];
val it = "i" : string
- middle [1, 2, 3, 4];
val it = 3 : int
- middle [1, 2, 3, 4, 5];
val it = 3 : int
- middle [1.0, 2.76, 3.14];
val it = 2.76 : real
```

   (a) (7 Pontos) Escreva a implementação de `middle`. Caso você queira, fique à vontade para usar a operação `div`, ex.:

```
- op div ;
val it = fn : int * int -> int
```

   (b) (3 Pontos) O tipo de `middle` é polimórfico. Que tipo de polimorfismo é esse, usado em `middle`?

4. Esta questão refere-se ao programa abaixo, implementado em `Gnu C`. Note que este programa não é C padrão.

```
sumSqs (int n) {
  int mul(int z) { return n * z; }

  if (n > 1)
    return mul(n) + sumSqs(n - 1);
  else
    return 0;
}

int main() {
  printf("%d\n", sumSqs(3));
}
```

(a) (2 Pontos) Qual o valor que esse programa imprime na saída padrão?

(b) (3 Pontos) Quais informações são necessárias no registro de ativação que o compilador de `Gnu C` precisa produzir para que o programa acima funcione corretamente? Responda genericamente. Por exemplo, o registro de ativação das funções deve conter variáveis locais. Além de variáveis locais, que outros tipos de informação devem estar presentes?

(c) (3 Pontos) Em C padrão, funções aninhadas não são permitidas. Essa restrição permite simplificar o registro de ativação das funções. Quais informações poderiam ser removidas do registro de ativação, nesse caso?

(d) (2 Pontos) Imagine que C padrão deixasse de suportar chamadas recursivas de funções. Isso seria, por exemplo, equivalente a marcarmos todas as variáveis, locais ou globais, com a palavra reservada `static`. Isso permitiria simplificar ainda mais o registro de ativação das funções. Nesse caso, quais simplificações seria possível fazer?