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. Esta questão refere-se ao programa abaixo, escrito em C:

```
int main(int argc, char** argv) {
  int *j = argv[1];
  printf("j = %d\n", j);
  printf("*j = %d\n", *j);
}
```

(a) (3 Pontos) Ao ser compilado e executado, o programa acima produz um erro do tipo "falha de segmentação", conforme os comandos reportados logo abaixo:

```
$> gcc segFault.c
$> ./a.out
j = 0
Segmentation fault
```

O que está errado no programa acima? Em outras palavras, qual o erro lógico no texto do programa que causa a falha de segmentação?

(b) (3 Pontos) Explique o que é uma **falha de segmentação**. A sua explicação deve conter uma resposta para a seguinte pergunta: esse tipo de erro é causado por código criado pelo compilador, durante a chamada `gcc segFault.c`; ou ele é criado pelo sistema operacional; ou ele é criado pela arquitetura?

(c) (4 Pontos) O programa abaixo é uma cópia do programa original, desta vez escrito em Java:

```
public class SegFault {
  public static void main(String argv[]) {
    String j = argv[1];
    System.out.println("*j = " + j);
  }
}
```

A chamada abaixo pode causar uma falha de segmentação? Sua resposta deve conter uma justificativa.

```
$> javac SegFault.java
$> java SegFault
```

2. Considere a função abaixo, escrita em SML:

```
fun s f = f f;
```

Se tentássemos declarar essa função, tal qual ela foi escrita, obteríamos o seguinte erro:

```
Error: operator is not a function [circularity]
  operator: 'Z
  in expression:
    f f
```

Por que não é possível declarar a função `s`? Dica: a solução para esta pergunta está relacionada ao sistema de tipos de SML, e os tipos envolvidos nessa declaração.

3. Números de Church podem ser implementados em SML da seguinte forma:

```
val ZERO = fn s => fn z => z;
val ONE = fn s => fn z => s z;
val TWO = fn s => fn z => s (s z);
val THREE = fn s => fn z => s (s (s z))
...
```

As próximas questões estão relacionadas à essa forma de implementar os Números de Church.

(a) (2 Pontos) Qual o tipo que o interpretador de SML inferiria para a função `ZERO`?

(b) (2 Pontos) Qual o tipo que o interpretador de SML inferiria para a função `ONE`?

(c) (3 Pontos) Implemente uma função `int2church`, que receba como entrada um inteiro n, e produza o número de Church c que corresponda a n. O tipo de sua função será `int -> ('a -> 'a) -> 'a -> 'a`. Sinta-se livre para utilizar a função sucessor, definida abaixo:

```
val SUCC = fn w => fn y => fn x => y(w y x)
```

Se o inteiro de entrada for menor que zero, então sua implementação deve retornar o número de Church `ZERO`. Note que a sua função deve **terminar** para qualquer entrada.

(d) (3 Pontos) Implemente uma função `intL2churchL`, de tipo `int list -> (('a -> 'a) -> 'a -> 'a) list`, que receba uma lista de inteiros $[n_1, n_2, \ldots, n_k]$ e produza uma lista $[c_1, c_2, \ldots, c_k]$, sendo cada c_i o número de Church que corresponde ao inteiro n_i, $1 \leq i \leq k$.

4. (10 Pontos) Em C++ o polimorfismo paramétrico é chamado de *templates*. Java também possui esse tipo de polimorfismo. Nesta linguagem, o polimorfismo paramétrico é chamado de *generics*. Abaixo temos dois programas, muito parecidos, um implementado em C++, e o outro implementado em Java:

C++

```
#include <vector>

class C {
  public:
    int i;
};

int main () {
  std::vector<int> vi;
  for (int i = 0; i < 10; i++) {
    vi.push_back(i);
  }
  for (int i = 0; i < vi.size(); i++) {
    printf("-> %d\n", vi[i]);
  }

  std::vector<C> vc;
  for (int i = 0; i < 10; i++) {
    C c;
    c.i = i;
    vc.push_back(c);
  }
  for (int i = 0; i < vc.size(); i++) {
    printf("-> %d\n", vc[i].i);
  }
  return 0;
}
```

Enquanto **aqui** inserimos o tipo primitivo diretamente, **aqui** ele é antes transformado em uma instância de Integer, que é um tipo composto.

Java

```
import java.util.Vector;

class C {
 public int f;
}

public class Templates {
  public static void main (String args[]) {
    Vector<Integer> vi = new Vector<Integer>();
    for (int i = 0; i < 10; i++) {
      vi.add(i);
    }
    for (int i = 0; i < vi.size(); i++) {
      System.out.println("-> " + vi.get(i));
    }

    Vector<C> vc = new Vector<C>();
    for (int i = 0; i < 10; i++) {
      C c = new C();
      c.f = i;
      vc.add(c);
    }
    for (int i = 0; i < vc.size(); i++) {
      System.out.println("-> " + vc.get(i).f);
    }
  }
}
```

Em C++ é possível termos estruturas polimórficas (*templates*) que acomodem tipos primitivos, como `vc` no programa acima. Em Java, por outro lado, tal não é possível. Uma estrutura polimórfica (*generic*), recebe somente referências para objetos. Assim, no programa Java acima, quando escrevemos `vi.add(i)`, sendo `i` um tipo primitivo, temos o chamado "encaixotamento"(*boxing*) de `i`. Em outras palavras, um objeto do tipo `Integer` é criado para comportar a variável `i`. É esse objeto, e não a própria variável `i`, que termina sendo inserida no vetor polimórfico. Note que em Java não é nem mesmo possível declararmos um vetor polimórfico parametrizado por um tipo primitivo, como `Vector<int>`, por exemplo.

Nesta questão você deve explicar a que se deve esta restrição da linguagem Java [1]. A resposta está relacionada à forma como *templates* e *generics* são implementados em cada linguagem.

[1] A linguagem não suporta polimorfismo paramétrico sendo o parâmetro um tipo primitivo. Somente tipos compostos, isto é, referências, podem ser parâmetros.

5. Esta questão refere-se à função `halve`, implementa em SML, e mostrada logo abaixo:

```
fun halve nil = (nil, nil)
  | halve [a] = ([a], nil)
  | halve (a::b::cs) =
    let
      val (x, y) = halve cs
    in
      (a::x, b::y)
    end
```

(a) (1 Pontos) qual o tipo da função `halve`?

(b) (1 Ponto) qual o resultado de `halve [1, 2, 3]`?

```
- halve [1, 2, 3] =
```

(c) (8 Pontos) Prove que se `L` for uma lista, então `halve L` produz duas listas, `L1` e `L2`, sendo que `length L = length L1 + length L2`. Sua prova deve ser simples, porém formal.

6. As funções desta questão devem ser escritas sem que sejam utilizadas as funções `foldr`, `foldl` e `map`.

   (a) (3 Pontos) Defina a função `mymap`, que tenha o mesmo tipo e comportamento de `map`. Por exemplo:

```
- mymap (fn x => x * x) [2, 3, 4, 5];
val it = [4,9,16,25] : int list
- mymap (fn x => x ^ ".") ["Viver nao eh preciso", "Navegar eh preciso"];
val it = ["Viver nao eh preciso.","Navegar eh preciso."] : string list
```

   (b) (2 Pontos) Qual o tipo de `mymap`?

   (c) (3 Pontos) Defina a função `myfoldr`, que tenha o mesmo tipo e comportamento de `foldr`. Por exemplo:

```
- myfoldr (op +) 0 [2, 3, 4, 5];
val it = 14 : int
- foldr (fn(x, y) => x ^ " " ^ y) "" ["a", "lingua", "eh", "minha", "patria"];
val it = "a lingua eh minha patria " : string
```

   (d) (2 Pontos) Qual o tipo de `myfoldr`?