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): em que ano foi decretada a abolição da escravidão no Brasil?

1. Esta questão refere-se ao programa abaixo, implementado em uma linguagem de programação hipotética, cuja sintaxe é baseada em Python:

```
def f(x) = x + 2
def h(g) = g(false)
h(f)
```

Explique como seria o sistema de tipagem da linguagem hipotética considerando cada um dos casos de tentativa de execução do programa acima. Especificamente, identifique se o sistema de tipagem é estático ou dinâmico e se há presença de algum dentre os seguintes tipos de polimorfismo: coerção, sobrecarga, paramétrico e/ou subtipagem. Justifique suas respostas.

(a) (2 Pontos) O programa não compila com o erro:

```
$> Linha 2. Tipo esperado: 'int'. Tipo encontrado: 'bool'
```

(b) (3 Pontos) O programa compila, mas durante a execução ele termina de modo anormal, com a mensagem:

```
$> Atribuição inválida. Tipo esperado: 'int * int'. Tipo encontrado: 'bool * int'
```

(c) (2 Pontos) O programa original compila, e retorna o valor 2. Porém, o programa abaixo não compila:

```
def f(x) = x + 2
def h(g) = g(f)
h(f)
```

(d) (3 Pontos) O programa original compila e retorna o valor 2. O programa abaixo também compila:

```
def f(x) = x + 2
def h(g) = g(f)
h(f)
```

Contudo, o programa acima termina de forma anômala, com a mensagem:

```
$> Operador +: int*int -> int não definido para type<fun>.
```

2. Esta questão refere-se a um experimento reportado no artigo "Towards a Green Ranking for Programming Languages", de autoria de Couto *et al.*, publicado no Simpósio Brasileiro de Linguagens de Programação de 2017. O principal resultado do artigo foi a tabela abaixo, que normaliza diferentes linguagens de programação com relação ao tempo que programas escritos nelas gastam para terminar, e a quantidade de energia do processador que eles consomem:

Total			
	Energy	Time	
C	1.00	1.00	C
Java	1.68	1.65	Java
Ocaml$\downarrow_1$	2.13	2.44	C#$\downarrow_2$
Fortran$\downarrow_2$	2.20	2.48	Ocaml$\uparrow_1$
C#$\uparrow_2$	2.21	2.63	Go$\downarrow_1$
Go$\uparrow_1$	3.04	3.91	Fortran$\uparrow_2$
Racket	7.35	10.29	Racket
Lua$\downarrow_2$	39.54	44.68	Jruby$\downarrow_1$
Jruby$\uparrow_1$	45.35	68.45	Perl$\downarrow_1$
Perl$\uparrow_1$	84.89	77.10	Lua$\uparrow_2$

(a) (2 Pontos) Duas das linguagens acima são consideradas funcionais. Quais?

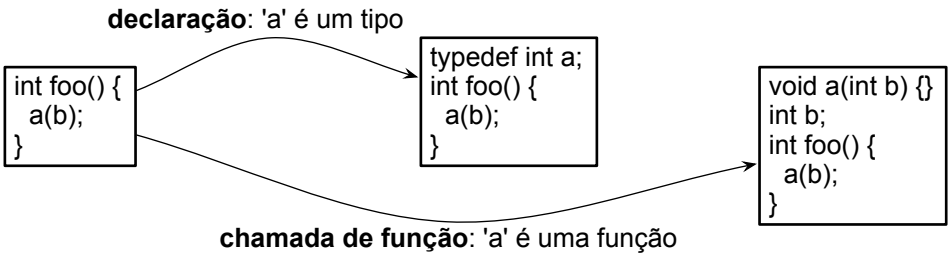
(b) (2 Pontos) As linguagens C, Ocaml, fortran e Go são compiladas para código de máquina. Cite duas razões que expliquem o fato de C ser muito mais rápida do que as outras três.

(c) (2 Pontos) Sendo Java uma linguagem virtualizada, era de se esperar que programas escritos em Java fossem mais lentos do que programas escritos em Fortran. Porém, isso não ocorre. Cite duas razões que permitam a uma linguagem virtualizada ser mais eficiente que uma linguagem compilada.

(d) (2 Pontos) Racket, Jruby, Perl e Lua foram as linguagens mais lentas (posto que seus programas demoraram mais para terminar). Qual a característica comum dessas linguagens que as tornam mais lentas?

(e) (2 Pontos) A razão (energia/tempo) não é constante entre as linguagens. Para C, essa razão é 1.00. Para Lua, ela é 0.51. Para Java, ela é 1.02. Isso leva a crer que Lua gasta menos energia por tempo de computação, e Java gasta mais. Cite duas razões que poderiam levar a este comportamento.

3. A gramática da linguagem C não é livre de contexto. Por exemplo, declarações determinam decisões de parsing. Para ilustrar essa característica da linguagem C, considere o programa abaixo:



Neste exemplo, `a(b)` pode ser tanto uma declaração de variável quanto uma chamada de função. O que determina qual regra de produção deve ser usada é a declaração de `a`.

(a) Abaixo são dados três outros tipos de programas ambíguos. Para cada programa, explique as duas naturezas sintáticas que o nome `a` pode assumir. Para cada resposta, escreva um pequeno código que, quando inserido antes da função `main`, permite que o programa compile. Um exemplo aparece logo acima.

- (2 Pontos)

```
int main() {
  a * b;
}
```

- (2 Pontos)

```
int main() {
  (a)-b;
}
```

- (2 Pontos)

```
int main() {
  (a)*b;
}
```

(b) (4 Pontos) Mesmo com esse tipo de ambiguidade, o parser de `gcc` ou de `clang` consegue analisar programas escritos em C ao custo de somente uma passada sobre o texto desses programas. O natural seria que fossem feitas duas passadas: a primeira coleta todas as declarações, e a segunda resolve as ambiguidades a partir daquela informação coletada anteriormente. Como é possível fazer parsing de C via uma passada somente?

4. Nesta questão você deverá implementar uma função, em ML, para determinar quando duas listas são palíndromos.

   (a) (2 Pontos) Comece implementando uma função `rev`, cujo tipo é: `'a list -> 'a list`. `rev` recebe uma lista, e produz o seu inverso:

   ```
   - rev [1, 2, 3];
   val it = [3,2,1] : int list
   - rev [1.0, 2.0, 3.0];
   val it = [3.0,2.0,1.0] : real list
   - rev [];
   ```

   (b) (1 Ponto) Qual a complexidade assimptótica da função `rev` criada na questão anterior?

   (c) (3 Pontos) Escreva uma função `zip`, cujo tipo é `'a list * 'b list -> ('a * 'b) list`. Essa função combina duas listas em uma lista de pares:

   ```
   - zip ([1, 2, 3], [true, false]);
   val it = [(1,true),(2,false)] : (int * bool) list
   - zip (["oi", "mundo"], [3.14, 15.92, 65.359]);
   val it = [("oi",3.14),("mundo",15.92)] : (string * real) list
   ```

   (d) (4 Pontos) Escreva a função palíndromo, cujo tipo é: `''a list -> bool`. Essa função recebe uma lista `L`, e retorna `true` se `L` for um palíndromo. Por exemplo:

   ```
   - palin [1, 2, 1];
   val it = true : bool
   - palin [true, false, false, true];
   val it = true : bool
   - palin [true, false, false];
   val it = false : bool
   - palin [];
   val it = true : bool
   - palin [1];
   val it = true : bool
   - palin [1, 2];
   val it = false : bool
   ```

   Observações:

   - você não pode simplesmente comparar a lista com seu inverso. A seguinte solução não é válida:

     `fun palin L = L = (rev L)`

   - Sua solução precisa, necessariamente, usar as seguintes funções: `rev`, `zip`, `map` e `foldr`.