Prova Final de Linguagens de Programação
- DCC024 -
Sistemas de Informação

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.
- Lembre-se do ganhador do Oscar de melhor filme em 2004. E lembre-se também: perguntando qual foi o filme ganhador você perde a sua pergunta.
- 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. A linguagem C é notória por apresentar problemas relacionados ao gerenciamento de memória. Qual o problema com cada um dos programas abaixo?

   (a) (3 pontos)

```
#include <stdio.h>
#include <stdlib.h>
void problem() {
  int* i = (int*) malloc (sizeof(int));
  *i = 3;
  printf("%d\n", *i);
}
int main() {
  problem();
}
```

   (b) (3 pontos)

```
#include <stdio.h>
#include <stdlib.h>
void dangling() {
  int* i = (int*) malloc (sizeof(int));
  int* j;
  *i = 3;
  free(i);
  j = (int*) malloc (sizeof(int));
  *j = 8;
  printf("%d\n", *i);
}
int main() {
  dangling();
}
```

   (c) (4 pontos)

```
#include <stdio.h>
unsigned* getMem() {
  unsigned x[10];
  int i;
  for (i = 0; i < 10; i++) {
    x[i] = i;
  }
  return x;
}
int main () {
  unsigned *p1 = getMem();
  unsigned *p2 = getMem();
  printf("[%u] = %d; [%u] = %d\n", p1, *p1, p2, *p2);
  printf("p2 = %u\n", p2[0]);
  p1[0] = 2;
  printf("p2 = %u\n", p2[0]);
}
```

2. As linguagens de programação usualmente fornecem aos desenvolvedores algum mecanismo para a implementação de *Tipos Abstratos de Dados*, ou TADs. TADs são tipos de dados definidos não pelos elementos em si, mas pelas propriedades que estes elementos possuem.

   (a) (4 pontos) Existe um princípio de programação que diz que tipos abstratos de dados deveriam ter os detalhes de implementação *encapsulados*. Por que o encapsulamento é importante?

   (b) (3 pontos) Um outro princípio reza que tipos abstratos de dados deveriam ser *extensíveis*, isto é, idealmente o desenvolvedor deveria ser capaz de estender um TAD com novas propriedades. A linguagem Python disponibiliza um mecanismo de extensão. Como se dá a extensão de TADs em Python?

   (c) (3 pontos) Complete a tabela abaixo com *verdadeiro* (V) e *falso* (F). Esta tabela distingue o suporte que diferentes linguagens de programação dão ao desenvolvimento de TADs. Use V quando a linguagem disponibilizar alguma sintaxe própria para o encapsulamento ou a extensão de TADs, e F doutro modo.

	Encapsulamento	Extensão
C	(F)	
SML		
Python		(V)
Java		

3. Esta questão refere-se à *busca binária*, um algoritmo de pesquisa baseada no princípio da divisão e conquista que possui complexidade $O(\ln n)$.

   (a) (5 pontos) A função `find` abaixo, escrita em C, busca o elemento `n` em um arranjo de inteiros `l` de tamanho `s`. Implemente este mesmo algoritmo em Python – lembre-se, a sintaxe correta é importante. A função `main` é apenas um exemplo de uso, e você não precisa implementá-la.

```
int find(int* l, int s, int n) {
  if (s) {
    int left = 0;
    int right = s - 1;
    while (1) {
      int index = (left + right) / 2;
      if (l[index] == n) {
        return 1;
      } else if (left >= right) {
        return 0;
      } else if (l[index] < n) {
        left = index + 1;
      } else {
        right = index - 1;
      }
    }
  }
  return 0;
}
```

```
int main() {
  int l[2] = {2, 3};
  printf("T0: %d\n", find(l, 2, 2));
  printf("T1: %d\n", find(l, 2, 3));
  printf("T2: %d\n", find(l, 2, 4));
  printf("T3: %d\n", find(NULL, 0, 1));
}
```

   (b) (5 pontos) Explique, em alto nível, como você faria para implementar um algoritmo similar em SML. O seu algoritmo deve permitir que sejam encontrados inteiros em uma estrutura de pesquisa em tempo logarítmico.

4. Exceções são mecanismos que várias linguagens de programação provêem para o tratamento de erros. Em Python, exceções são definidas como classes. Por exemplo, a classe `ArithmeticException` abaixo representa um tipo de exceção particular:

```
class ArithmeticException(Exception):
  def __init__(self, msg):
    self.value = msg
  def __str__(self):
    return repr(self.value)
```

(a) (4 pontos) Implemente uma função `div` em Python, que receba dois números, `n` e `d` e retorne o quociente `n/d`. A sua implementação deve disparar uma exceção do tipo `ArithmeticException` com a mensagem ‘‘`Denominador igual a zero.`", caso o valor de `d` seja de fato zero.

(b) (6 pontos) Considere o programa abaixo, que testa a função `div`:

```
while True:
  n = float(raw_input("Informe o dividendo: "))
  d = float(raw_input("Informe o divisor: "))
  if d == 0:
    break
  r = div(n, d)
  print "Resultado = ", r
```

Este programa termina assim que o usuário informa um valor de `d` igual a zero. Modifique esta implementação para tratar dois tipos de exceção:

- `ValueError`: se os caracteres fornecidos na entrada não definirem um número em formato válido. Neste caso, simplesmente peça ao usuário que informe novos números.
- `ArithmeticException`: se o valor fornecido para `d` for zero. Neste caso, termine o *loop*.

5. Diga qual o resultado de cada uma das sentenças abaixo, escritas em Prolog. Os resultados possíveis são:

   - `sim` – Neste caso a unificação é possível. Você deve escrever qual o unificador mais geral que torna a sentença verdadeira.
   - `não` – Neste caso não existe um unificador mais geral para a sentença.
   - `erro` – Erros acontecem quando variáveis não foram suficientemente definidas.

   (a) (1 Ponto) `X = Y.`

   (b) (1 Ponto) `(a, X, Y) = (X, Y, b).`

   (c) (1 Ponto) `(a, X, X) = (Y, X, b).`

   (d) (1 Ponto) `X = 1 + 2 + 3.`

   (e) (1 Ponto) `X is 1 + 2 + 3.`

   (f) (1 Ponto) `X =:= 1 + 2 + 3.`

   (g) (2 Pontos) `E = [1, 2], E = [F|G].`

   (h) (2 Pontos) `X = 2, Y = 1+1, X = Y.`

6. Uma das principais estruturas de dados em Prolog são as listas. Resolva as duas questões abaixo usando listas:

   (a) (5 Pontos) Escreva um predicado `maxList(L, M)`, que receba uma lista `L` de números e unifique `M` com o maior número nesta lista. O predicado deve falhar se a lista estiver vazia.

   (b) (5 Pontos) Escreva um predicado `ordered(L)`, que seja verdadeiro se a lista `L` estiver em ordem crescente.