Segunda Prova de Linguagens de Programação
- DCC024 -
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. São perguntas: "*Posso fazer uma pergunta?*"ou "*Quanto tempo falta?*". Para a pergunta: "*Posso ir ao banheiro*", a resposta é *sim*.
- 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!

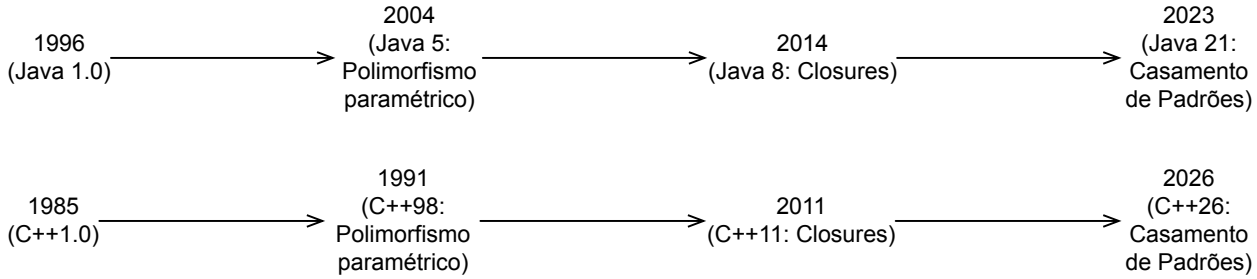
Tabela 1: Pontos acumulados (para uso do instrutor)

Questão 1	Questão 2	Questão 3	Extra

Questão Extra (0.5 Pontos): Em outubro de 2009, comemorando três anos de existência, a revista Rolling Stone Brasil publicou uma edição especial com as 100 maiores músicas brasileiras de todos os tempos. Cite uma música que está entre as top três.

1. Construção; 2. Águas de Março; 3. Carinhoso

1. Nas últimas duas décadas temos assistido a uma crescente incorporação de capacidades típicas de linguagens funcionais em linguagens imperativas. A figura abaixo ilustra essa evolução em Java e C++:



(a) (3 Pontos) Escreva um programa, em C++, que ilustre uma ocorrência de polimorfismo paramétrico.

```
int main() {
  std::vector<int> v;
}
```

```
template <class T>
T GetMax (T a, T b) {
  T result;
  result = (a>b)? a : b;
  return (result);
}
```

(b) (3 Pontos) Escreva um programa, em Python 3.0, que contenha um exemplo de um *closure*.

```
v = lambda x: lambda y: x + y
f = v(1) # f is a closure
f(2)
```

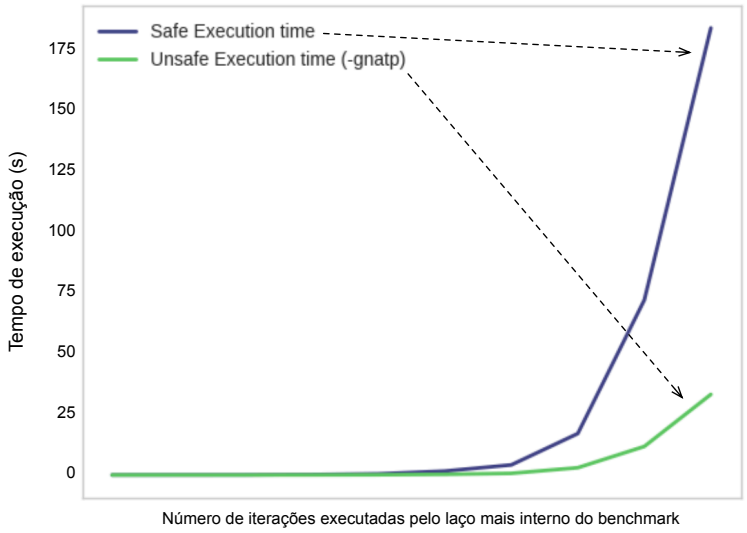
```
def make_incrementor(n):
  def inc(x):
    return x + n
  return inc
```

```
def make_incrementor(n):
  return lambda x: x + n
```

(c) (4 Pontos) Escreva um programa em SML/NJ que mostre um exemplo do uso de casamento de padrões.

```
fun f nil = 0
  | f (true::rest) = 1 + f rest
  | f (false::rest) = f rest;
```

2. Ada é uma linguagem de programação compilada, fortemente tipada, projetada para desenvolver sistemas confiáveis, seguros e de alto desempenho por meio de verificações rigorosas em tempo de compilação e de execução. Ada pode ser compilada em modo seguro, com verificações automáticas de erros em tempo de execução (como limites de vetores e estouro aritmético), ou em modo inseguro, no qual essas verificações são desabilitadas para maximizar o desempenho. Essa segurança tem um impacto grande na eficiência de programas, conforme mostra a figura abaixo:



SAFE ADA insere verificações para:

* Limites de arranjos
* Overflow de aritmética
* Verificação de ponteiro nulo
* Coerções entre classes

(a) (2 Pontos) O que seria um "Estouro" ou (*overflow*) aritmético?

Um estouro aritmético (arithmetic overflow) ocorre quando o resultado de uma operação matemática excede a capacidade máxima de armazenamento do tipo de dado em uso na memória do computador.

(b) (2 Pontos) O que ocorre quando um programa Java tenta invocar um método sobre um objeto cuja referência é nula?

Uma exceção: NullPointerException (não é necessário informar o nome da exceção)

(c) (2 Pontos) O que ocorre quando um programa C++ tenta invocar um método sobre um objeto apontado por um ponteiro nulo?

Ocorre um comportamento indefinido.

(d) (2 Pontos) É possível acessar um vetor fora dos limites alocados em um programa Python?

Não. Se você tentar ler ou atribuir um valor a um índice inexistente (maior que o tamanho ou negativo fora do escopo), o Python interromperá a execução imediatamente e lançará uma exceção

(e) (2 Pontos) O que seria uma coerção insegura entre `structs` na linguagem C?

Uma coerção insegura (ou unsafe cast) entre structs na linguagem C ocorre quando você força o compilador a tratar o bloco de memória de uma estrutura de dados como se fosse de outro tipo de estrutura.

3. Considere a soma abaixo:

```
  SEND
+ MORE
------
 MONEY
```

Cada letra representa um dígito decimal **distinto** entre 0 e 9. Letras iguais representam o mesmo dígito, e letras diferentes representam dígitos diferentes. Além disso, os números `SEND` e `MORE` não podem começar com o dígito zero. O programa abaixo determina quais dígitos devem ser atribuídos às letras S, E, N, D, M, O, R e Y para que a soma seja correta:

```
puzzle([S,E,N,D,M,O,R,Y]) :-
    Digits = [                                ]
    perm(Digits, [S,E,N,D,M,O,R,Y,_,_]),
    S \= 0,
    M \= 0,
    SEND  is 1000*S + 100*E + 10*N + D,
    MORE  is 1000*M + 100*O + 10*R + E,
    MONEY is 10000*M + 1000*O + 100*N + 10*E + Y,
    SEND + MORE =:= MONEY.
```

C

D

A

```
?- select(E, [a,b,c], L).
E = a,
L = [b, c] ;
E = b,
L = [a, c] ;
E = c,
L = [a, b].
```

B

```
?- perm([a,b,c], X).
X = [a, b, c] ;
X = [a, c, b] ;
X = [b, a, c] ;
X = [b, c, a] ;
X = [c, a, b] ;
X = [c, b, a] ;
```

(a) (3 Pontos) Existe um predicado `select(E, S, L)` na biblioteca padrão de Prolog. Esse predicado é verdade quando `S` é uma lista que contém pelo menos uma ocorrência do elemento `E`, e `L` é a mesma lista, porém sem essa ocorrência. Veja o exemplo (A) na figura acima. Implemente esse predicado.

```
select(X, [X|L], L).
select(X, [H|L], [H|LR]) :- select(X, L, LR).
```

(b) (3 Pontos) O predicado `puzzle` acima usa um subpredicado `perm(L, X)`, que é verdade quando a lista `X` é uma permutação da lista `L`. Veja o exemplo (B) na figura acima. Implemente esse predicado. Você **deve** usar o predicado `select` implementado na questão anterior.

```
perm([], []).
perm(T, [H|LL]) :- select(H, T, L1), perm(L1, LL).
```

```
perm([], []).
perm([H|T], L) :- perm(T, Lx), select(H, L, Lx).
```

(c) (2 Pontos) Como você deveria inicializar a variável `Digits` na parte (C) da figura acima?

`Digits =` **[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]**

(d) (2 Pontos) O predicado `puzzle` contém a igualdade `SEND + MORE =:= MONEY`. Por que é necessário usar o predicado `=:=` em vez da unificação simples `=`, ou do operador "`is`"?

Porque o lado esquerdo precisa ser avaliado aritméticamente