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**.

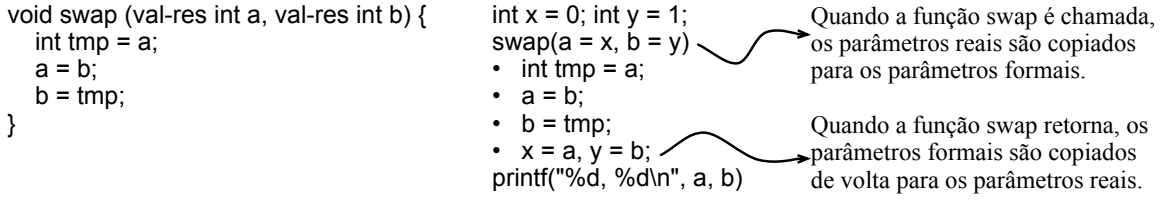
Alguns conselhos:

- 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. Existe um mecanismo de passagem de parâmetros chamado *passagem por valor-resultado*. De acordo com esse mecanismo, os parâmetros reais são avaliados, e copiados para os parâmetros formais logo que a função é chamada. Quando a função termina, os valores dos parâmetros formais são copiados de volta para os parâmetros reais. A figura abaixo ilustra essa semântica:



(a) (1 Ponto) O que será impresso pela chamada `printf` vista acima?

(b) (9 Pontos) Assuma que a linguagem C possua chamadas por valor-resultado, e por referência. Parâmetros passados por valor-resultado devem ser prefixados com a palavra chave `val-res`. Parâmetros passados por referência devem ser prefixados com a palavra chave `ref`, conforme visto nos programas logo abaixo:

Passagem por valor-resultado:

```
void max (int x, int y, val-res int r) {
   r = x;
   if (y > x) {
      r = y;
   }
}
```

Passagem por referência:

```
void max (int x, int y, ref int r) {
   r = x;
   if (y > x) {
      r = y;
   }
}
```

Do ponto de vista prático, existe alguma diferença semântica entre essas duas formas de passagem de parâmetros? Em caso negativo, explique porque elas sempre geram resultados iguais. Em caso positivo, demonstre a sua resposta com um trecho de código nessa linguagem C estendida. Escreva a implementação de sua função, explique quais parâmetros são passados de quais formas, mostre o código da chamada, e escreva o resultado esperado assumindo passagem por referência e por valor-resultado.

2. Essa questão refere-se à função `traverse` abaixo, implementada em Python:

```
def traverse(v):
  if len(v) >= 1:
    print(v[0])
  if len(v) > 1:
    traverse(v[1])
```

(a) (2 Pontos) A função `traverse` possui cauda rasa ou não? Justifique a sua resposta.

(b) (4 Pontos) Considere as duas chamadas à função `traverse` feitas na figura abaixo. Em (a) a função termina, e imprime "1"duas vezes. Em (b) a função não termina. Qual a diferença entre as duas estruturas, `a` e `b`, que justifiquem esse comportamento?

(a)
```
>>> a = [1]
>>> a = [1, a]
>>> traverse(a)
1
1
```

(b)
```
>>> b = [1]
>>> b.append(b)
>>> traverse(b)
...
```

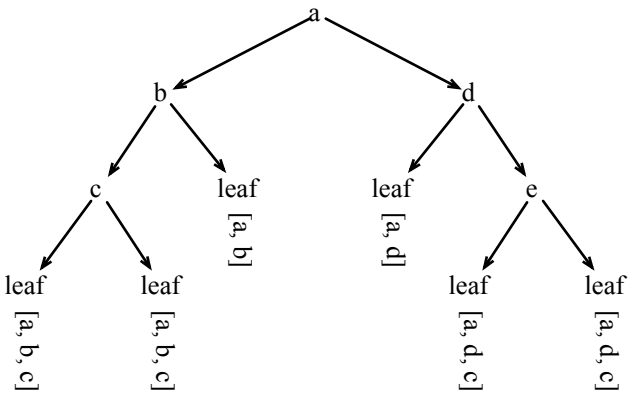
(c) (2 Pontos) Considere um subconjunto de SML que possua somente tuplas, listas, tipos algébricos, funções (mas não *closures*) e os cinco tipos primitivos: `int`, `real`, `bool`, `char` e `string`. Como você poderia construir uma estrutura como aquela vista em 2.b(a)? Se tal não for possível, você deve justificar a sua resposta. Doutro modo, você deve escrever o código que constrói aquela estrutura.

(d) (2 Pontos) Responda a mesma questão que 2.c, mas desta vez com relação à estrutura `b`, criada com os comandos `b = [1]; b.append(b)`.

3. (10 Pontos) Definimos uma árvore em Prolog via dois predicados. `node(V, T1, T2)` representa um nodo da árvore que guarda o valor `V`, e possui sub-árvores `T1` e `T2`. E `leaf` representa um nodo folha da árvore. A título de exemplo, os predicados abaixo somam uma árvore de inteiros:

```
sum(node(V, T1, T2), VV) :- sum(T1, V1), sum(T2, V2), VV is V + V1 + V2.
sum(leaf, 0).
```

Nesta questão você deve criar um predicado `search(T, L)`, que seja verdade quando existir um caminho em T que seja equivalente à lista `L`. Por exemplo, esse predicado retorna seis respostas para a árvore `node(a, node(b, node(c, leaf, leaf), leaf), node(d, leaf, node(e, leaf, leaf)))`, conforme pode ser visto na figura abaixo:




```
search(node(a, node(b, node(c,
leaf, leaf), leaf), node(d, leaf,
node(e, leaf, leaf))), L).

L = [a,b,c] ? ;

L = [a,b,c] ? ;

L = [a,b] ? ;

L = [a,d] ? ;

L = [a,d,e] ? ;

L = [a,d,e]
```

4. Esta questão refere-se ao algoritmo *merge*, escrito em Portugol logo abaixo:

```
merge (vetor de inteiros ℓ1, vetor de inteiros ℓ2)
  seja s1 o tamanho de ℓ1 e seja s2 o tamanho de ℓ2
  seja ℓ um vetor vazio de tamanho s1 + s2
  seja i, i1 e i2 inteiros iguais a zero
  enquanto i for menor que a soma (s1 + s2) faça
    ℓ[i] = ℓ1[i1]
    i1 = i1 + 1
    i = i + 1
    ℓ[i] = ℓ2[i2]
    i2 = i2 + 1
```

(a) (3 Pontos) Escreva esse algoritmo em SML. Perceba que SML – a parte vista em aula – não possui estruturas de acesso aleatório. Assim, você terá de implementar um algoritmo que tenha propósito semelhante à *merge*, mas que use listas.

(b) (3 Pontos) Implemente o mesmo algoritmo *merge*, mas desta vez em Prolog. Assim como na questão anterior, use listas para representar vetores.

(c) (4 Pontos) Agora, implemente aquele algoritmo em Python.

5. Essa questão refere-se ao programa abaixo, que foi implementado na linguagem de programação C++:

```
1  int foo(int n) {
2    if (n < 2) {
3      return 1;
4    } else {
5      const int x = n - 1;
6      printf("Endereco = %u\n", &x);
7      return n * foo(x);
8    }
9  }
```

```
10 int main() {
12   foo(3);
13 }
```

(a) (3 Pontos) O endereço que é impresso na linha 6 é sempre o mesmo em cada uma das duas chamadas de `foo` que ocorrerão nesse programa? Fundamente sua resposta com base na área de alocação: estática, pilha ou *heap* usada para armazenar a variável `x`.

(b) (2 Pontos) caso substituíssemos a linha 6 pelo comando `x = x - 1` o programa não estaria correto, pois estaríamos tentando atualizar uma constante. Quem faz a verificação ou prevenção desse erro? O compilador, o sistema operacional ou a arquitetura?

(c) (3 Pontos) Explique como o ator escolhido acima (compilador, sistema operacional ou arquitetura) impede que constantes sejam sobre-escritas em C++.

(d) (2 Pontos) Reescreva a função `foo` para que ela passe a ser uma função de cauda rasa.

6. Duas das mais antigas linguagens de programação ainda em uso hoje são Fortran e Lisp. Fortran apresenta um modelo de programação mais imperativo, enquanto Lisp tende ao lado mais declarativo dessa taxonomia. Hoje, a maior parte das linguagens de programação em largo uso são mais parecidas com Fortran que Lisp. Enquanto linguagens imperativas, elas permitem que programas sejam descritos como algoritmos. Algoritmos são sequências de passos que descrevem como o estado do computador deve ser alterado. Embora as linguagens declarativas nunca tenham alcançado a popularidade de suas irmãs imperativas, hoje podemos observar que várias linguagens de programação modernas herdaram algumas de suas características. Nesta questão revisaremos algumas dessas características.

   (a) (3 Pontos) Escreva um *trecho* de código em Java, que use algum conceito de linguagem de programação introduzido por alguma linguagem funcional.

   (b) (3 Pontos) Agora, escolha uma funcionalidade de Python que foi introduzida por Lisp, e ilustre-a com um trecho de código.

   (c) (2 Pontos) Escolha agora, uma segunda inovação de Lisp, diferente daquela escolhida na questão 6.b, e indique três linguagens diferentes que a usem. Note que a propriedade ou capacidade de Lisp escolhida nesta questão não pode estar presente em Fortran.

   (d) (2 Pontos) Finalmente, aponte uma terceira capacidade de Lisp (diferente daquelas vistas em 6.b e 6.c), também não presente em Fortran, que seja amplamente encontrada em linguagens de programação modernas. Indique três linguagens que façam uso desta característica de Lisp.