Primeira Prova de Linguagens de Programação
- DCC024W -
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.
- 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 Python:

```
def div(a, b):
  if b == 0:
    return "Nao sei dividir"
  else:
    return a / b

>>> div(40, 3)
13
>>> div(40, 0)
'Nao sei dividir'
```

Esse programa evidencia a *tipagem dinâmica* de Python.

(a) (2 Pontos) O que é tipagem dinâmica?

(b) (2 Pontos) Não é possível construir uma função exatamente como `div` em SML, pois essa linguagem é estaticamente tipada. Por que a tipagem estática é um empecilho para a construção de uma função como `div` em SML?

(c) (3 Pontos) Defina um tipo algébrico `RESULT` que seja a união de dois tipos, os inteiros e os *strings*. O tipo `RESULT` terá dois rótulos: `INT` associado aos inteiros, e `STR`, associado aos *strings*.

(d) (3 Pontos) Use o tipo algébrico `RESULT`, definido na questão anterior, para escrever uma função `sml_div`, cujo tipo seja `int * int -> RESULT`. Por exemplo:

```
- sml_div(3, 0);
val it = STR "Nao sei dividir" : RESULT
- sml_div(30, 4);
val it = INT 7 : RESULT
```

O operador de divisão inteira em SML é `div`.

2. Esta questão refere-se à função `partition`, cuja implementação em SML é dada logo abaixo:

```
fun partition (pivot, nil) = (nil,nil)
  | partition (pivot, first :: others) =
      let
        val (smalls, bigs) =
          partition(pivot, others)
      in
        if first < pivot
          then (first::smalls, bigs)
          else (smalls, first::bigs)
      end;
```

(a) (2 Pontos) Qual é o tipo da função `partition`? Assuma que o operador `<` possui tipo `int * int -> bool`.

(b) (8 Pontos) Se `L` é uma lista, e o resultado de `partition(_, L)` são duas listas `L1` e `L2`, então prove que o tamanho de `L` é igual à soma dos tamanhos de `L1` e `L2`.

3. Esta questão refere-se à função lógica `NAND`, também conhecida como "conjunção negada". Uma implementação desta função, em SML, produziria os seguintes resultados:

```
- NAND(true, true);
val it = false : bool
- NAND(true, false);
val it = true : bool
- NAND(false, true);
val it = true : bool
- NAND(false, false);
val it = true : bool
```

   (a) (4 Pontos) Implemente a função `NAND`, de tipo `bool * bool -> bool`, de forma que a sua implementação retorne os mesmos resultados que aqueles vistos na sessão de SML logo acima.

   (b) (6 Pontos) Alguns operadores binários, quando usados nas funções `foldr` e `foldl` retornam os mesmos resultados. Por exemplo:

```
- foldr (op +) 0 [1,2,3,4];
val it = 10 : int
- foldl (op +) 0 [1,2,3,4];
val it = 10 : int
```

   Contudo, há operadores que retornam resultados diferentes. Por exemplo:

```
- foldr (op ^) "" ["a", "b", "c"];
val it = "abc" : string
- foldl (op ^) "" ["a", "b", "c"];
val it = "cba" : string
```

A sua implementação de `NAND` retorna o mesmo resultado tanto para `foldl` quanto para `foldr`? Justifique a sua resposta.

4. Esta questão refere-se ao programa abaixo, implementado em C. Este programa utiliza funções aninhadas. Tais funções não são parte de ANSI C, mas o compilador `gcc` as suporta, se a opção `-fnested-functions` for passada em linha de comando.

```
1  #include <stdio.h>
2
3  int sum(int x, int limit) {
4    int prod(int begin) {
5      if (begin >= limit)
6        return 1;
7      else
8        return begin * prod(begin + 1);
9    }
10   return prod(x);
11 }
12
13 int main(int argc, char** argv) {
14   printf("sum(2, 1) = %d\n", sum(2, 1));
15 }
```

(a) (4 Pontos) Como é possível que a implementação da função `prod` tenha acesso ao valor da variável `limit`, sendo que essa variável não foi declarada no escopo de `prod`? Explique como o valor de `limit` pode ser encontrado a partir do registro de ativação de `prod`.

(b) (6 Pontos) Desenhe todos os registros de ativação ativos quando a linha 6 do programa acima é atingida pelo fluxo de execução. Procure lembrar-se de todas as informações que são armazenadas em registros de ativação. Dica: abaixo vê-se uma sessão do depurador `gdb`:

```
~/Fernando$ gcc -g -fnested-functions nested.c
~/Fernando$ gdb a.out
(gdb) break nested.c:6
Breakpoint 1 at 0x1f91: file nested.c, line 6.
(gdb) run
Breakpoint 1, prod (begin=2) at nested.c:6
(gdb) backtrace
#0  prod (begin=2) at nested.c:6
#1  0x00001f7c in sum (x=2, limit=1) at nested.c:10
#2  0x00001fd8 in main (argc=1, argv=0xbffff738) at nested.c:14
(gdb) quit
```

5. Esta questão refere-se à gramática lógica abaixo, implementada em Prolog, que reconhece listas de `a`'s:

```
list --> ['['], seq, [']'].
list --> ['['], [']'].

seq --> [a], seq.
seq --> [a].
```

(a) (2 Ponto) Desenhe uma possível árvore de derivação para a *string* `'[', a, a, a, ']'`.

(b) (1 Ponto) Sabendo que a gramática mostrada nesta questão não é ambígua, além da árvore que você desenhou no item anterior, quantas outras árvores de derivação existem para a *string* `['[', a, a, a, ']'`, no mínimo?

(c) (7 Pontos) Escreva uma nova gramática, inserindo atributos na gramática lógica mostrada acima. A nova gramática deve ser capaz de contar o número de elementos em listas. Como um exemplo, essa nova gramática deverá proceder como na sessão prolog abaixo:

```
?- list(N, ['[', a, a, a, ']'], []).
N = 3.

?- list(N, ['[', ']'], []).
N = 0.

?- list(N, ['[', a, ']'], []).
N = 1.
```

6. Dizemos que uma linguagem é segura quando esta linguagem não permite que operações sejam aplicadas a argumentos que não possuam os tipos previstos por estas operações. C e C++ são linguagens inseguras, pois muitas vezes valores armazenados em memória são utilizados sem qualquer fiscalização de seus tipos.

   (a) (5 Pontos) Escreva um programa em C ou C++ que evidencie o caráter inseguro de uma dessas linguagens.

   (b) (5 Pontos) Existem linguagens mais antigas que C ou C++ que são consideradas seguras, logo, a possibilidade de uso inseguro de tipos não é devido à ignorância sobre os perigos dessa abordagem. ML, por exemplo, já havia sido definida dez anos antes de C++, porém enquanto ML é uma linguagem considerada segura, C++ não é. Cite um fator que motivou o desenho inseguro de C++.