

# Trabalho de implementação 2

Resolvedor de Suguru em LISP

Isac de Souza Campos (17200449)  
Felipe de Campos Santos (17200441)

## O Problema

Neste segundo trabalho, o objetivo era refazer o algoritmo feito para resolver o Suguru em uma outra linguagem funcional. Optamos por LISP, por já estar usando esta na matéria e poder continuar desenvolvendo e praticando a mesma em paralelo com as atividades a serem entregues. Para isso, aproveitamos do algoritmo do Isac em Haskell e “traduzimos” ele para LISP, fazendo as alterações necessárias durante o caminho.

## As Entradas

Nossa modelagem de dados é representada por duas listas primariamente, aqui chamadas de “table design” e “table”, representando, respectivamente, as áreas do tabuleiro e os valores do tabuleiro.

A lista table design mostra as áreas separando elas por números, ou seja, todas as células que tem o número 1 fazem parte da área 1, todas as com número 2 fazem parte da área 2, e assim respectivamente.

Já a lista table traz, para cada célula, seu valor inicial. Aquelas que inicialmente estariam vazias são preenchidas com -1

O tabuleiro usado no código, retirado do site [Janko](#), é mostrado abaixo:

			3			2	
4							
	2						
	1	5			1	5	
	2						
				4			4
					3		
	5				5		

```

(setq table_design '(0 0 0 1 2 2 3 3
                     0 4 4 1 2 2 3 3
                     4 4 1 1 2 5 5 3
                     4 6 6 1 7 7 5 5
                     8 8 6 6 7 9 9 5
                     10 8 8 6 7 11 9 9
                     10 10 8 12 7 11 11 9
                     10 10 12 12 12 12 11 11
                     )
)

(setq table '(-1 -1 -1 3 -1 -1 2 -1
             4 -1 -1 -1 -1 -1 -1 -1
             -1 2 -1 -1 -1 -1 -1 -1
             -1 1 5 -1 -1 1 5 -1
             -1 2 -1 -1 -1 -1 -1 -1
             -1 -1 -1 -1 4 -1 -1 4
             -1 -1 -1 -1 -1 3 -1 -1
             -1 5 -1 -1 -1 5 -1 -1
             )
)

```

Para alterar a entrada, deve se alterar essas duas definições no código da maneira adequada.

## O Programa

O grande mantenedor desse programa é a função *checkValidation*, que faz uma varredura das células utilizando as tabelas acima como suporte e fazendo a “limpeza” dos arredores das células e também da área que elas se encontram, chamando as funções *isNeighborhoodClean*, *getBoxPositions* e *isBoxClean* para isso, até que seja terminada essa varredura.

```

(defun checkValidation (f_table updt_table n box_n v vf)
  (if (null f_table)
      (make-tuple :x NIL :y NIL)
      (progn
        (setq neighborhoodClean (isNeighborhoodClean updt_table n v))
        (setq boxPositions (getBoxPositions box_n boxes_positions))
        (setq boxClean (isBoxClean updt_table boxPositions v))
        (...))
  )
)

```

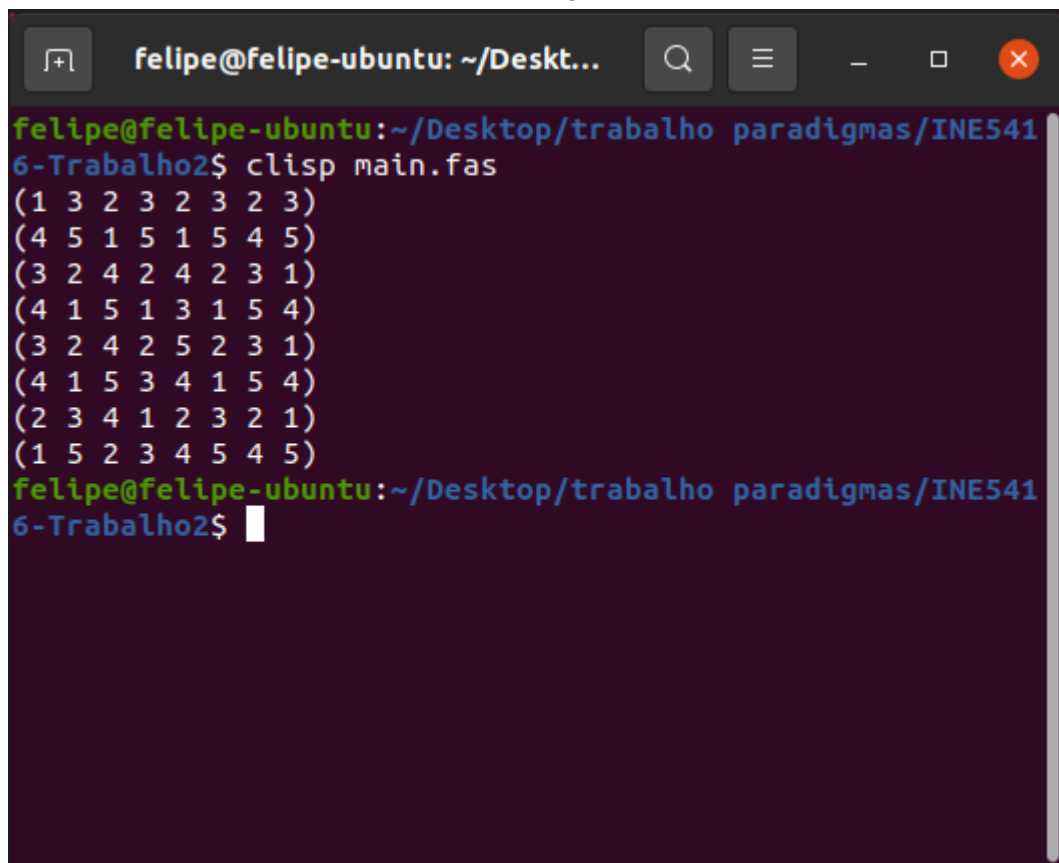
Em comparação com Haskell, Lisp é mais intuitivo na maneira de fazer operações, apesar da sintaxe guiada pelos parênteses ser bem desprazerosa de se trabalhar. Muitas vezes a própria IDE deletava um parêntese adicional quando nós deletávamos um outro parêntese, assim impedindo que fosse compilado e muitas vezes não apontando o erro nem perto de onde ele realmente tinha acontecido.

Outro problema que encontramos foi a falta de um list comprehension em LISP como existe em Haskell, mas para isso foi criada uma função auxiliar que recebe os elementos da lista a ser varrida e executa as operações de forma similar à o list comprehension. Também foi necessária a criação de uma estrutura que definisse uma tupla, dado que a criação de listas ou tuplas vazias em LISP retorna NIL, e não uma estrutura vazia.

```
;;type tuple
(defstruct tuple
  x
  y
)
```

## O Resultado

Com tudo isso, foi possível fazer a tradução completa do programa para LISP. Ao executar o arquivo main pelo terminal, obtemos o seguinte resultado:

A screenshot of a terminal window titled 'felipe@felipe-ubuntu: ~/Desk...'. The prompt is 'felipe@felipe-ubuntu:~/Desktop/trabalho paradigmas/INE5416-Trabalho2\$'. The user enters 'clisp main.fas'. The output consists of ten lines of tuples: (1 3 2 3 2 3 2 3), (4 5 1 5 1 5 4 5), (3 2 4 2 4 2 3 1), (4 1 5 1 3 1 5 4), (3 2 4 2 5 2 3 1), (4 1 5 3 4 1 5 4), (2 3 4 1 2 3 2 1), and (1 5 2 3 4 5 4 5). The prompt returns to 'felipe@felipe-ubuntu:~/Desktop/trabalho paradigmas/INE5416-Trabalho2\$' with a cursor.

E, comparando com o resultado dado pelo próprio site, confirmamos a validade do programa:

1	3	2	3	2	3	2	3
4	5	1	5	1	5	4	5
3	2	4	2	4	2	3	1
4	1	5	1	3	1	5	4
3	2	4	2	5	2	3	1
4	1	5	3	4	1	5	4
2	3	4	1	2	3	2	1
1	5	2	3	4	5	4	5

## Sobre o grupo

Na entrega anterior, o Isac havia feito o trabalho sozinho e o Felipe havia feito em um grupo no qual ele não sentiu pro-atividade e organização, por isso nessa segunda entrega decidimos nos juntar. Nossa comunicação foi bem fluida, e ambos foram pro-ativos para fazer o que precisava ser feito no momento, sem a necessidade de ficar separando passo a passo o que cada um faria ou ter uma cobrança.

O vídeo da apresentação pode ser acessado por esse [link](#).