

Ficha de Laboratório Nº 4: Funções recursivas, *Tracer* e Explorador de chamadas a funções

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal

Prof. Joaquim Filipe

Eng. Filipe Mariano

Objetivos da Ficha

- Praticar a sintaxe para a criação de funções em Lisp
- Funções recursivas recorrendo ao *Editor* do LispWorks
- Análise de chamadas de funções no LispWorks
- Introduzir recomendações de programação em Lisp

1. Exemplo de Definição de Funções Recursivas

1. Descarregue o ficheiro disponível no Moodle, chamado **laboratorio4.lisp** e guarde-o no seu computador.
2. Abra o ficheiro no IDE LispWorks.
3. Observe a estrutura de comentários no início do ficheiro para indicar o conteúdo do ficheiro e o seu autor.
4. Analise a função **comprimento** definida no ficheiro.
5. Teste a função no *Listener*, chamando-a tendo a lista **((3 4 5) 1 2))** como argumento de entrada.

```
CL-USER > (comprimento '(((3 4 5) 1 2) 3 4 5 6))  
5
```

Resultado: A função devolve o número de elementos da lista passada como argumento de entrada.

2. Visualizar Chamadas a uma Função

O LispWorks disponibiliza várias ferramentas úteis para a análise das chamadas e da relação entre funções implementadas em Lisp, e que serão resumidas nas secções seguintes.

2.1. Explorador de Chamadas a Funções

O *Function Call Browser* ajuda a analisar as interações entre uma dada função em Lisp e outras que são chamadas por ela, ou em que a mesma é usada. O exemplo seguinte ilustra a utilização desta funcionalidade no LispWorks.

1. No *Listener* ou no *Editor* avalie as funções contidas no ficheiro **laboratorio4.lisp**, de modo a que fiquem definidas no interpretador.
2. Abrir o *Function Call Browser*, clicando no ícone ilustrado na Figura 1. A janela deste módulo aparece tal como representada na Figura 2.
3. Nesta janela insira o nome da função **comprimento** no espaço denominado *Function* e pressione a tecla ENTER. O nome da função irá ser adicionado na zona "Called By" desta janela, com a indicação das funções que chamam **comprimento** (caso existam).
4. Seleccione a zona "Calls Into". A janela ficará com o aspecto apresentado na Figura 3, contendo a indicação das funções chamadas por **comprimento**.
5. É possível obter um resumo da definição da função, e das informações anteriormente indicadas, seleccionando a zona "Text".



Figura 1: O ícone do *Function Call Browser*.

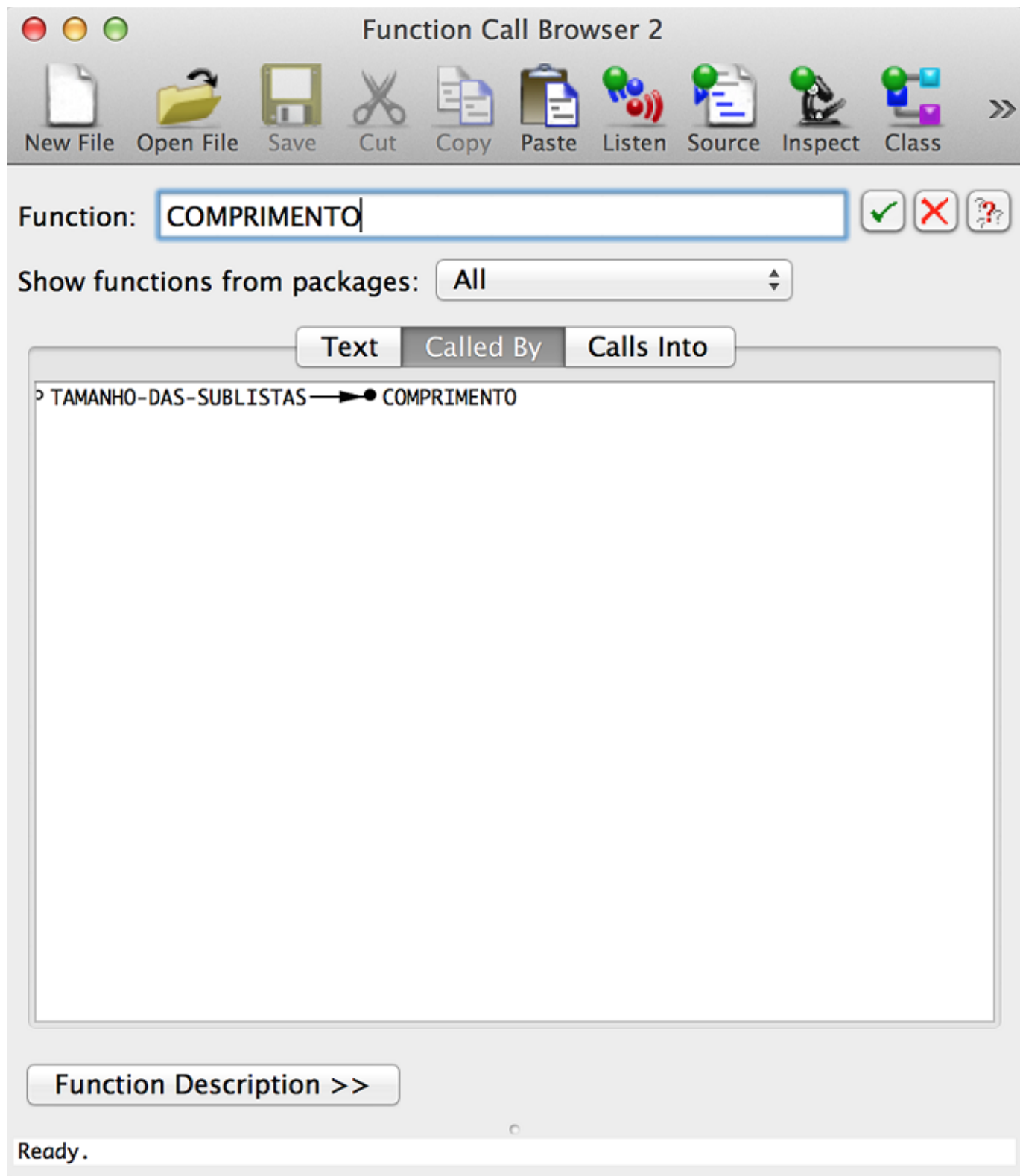


Figura 2: Resumo das funções nas quais a função `comprimento` é utilizada.

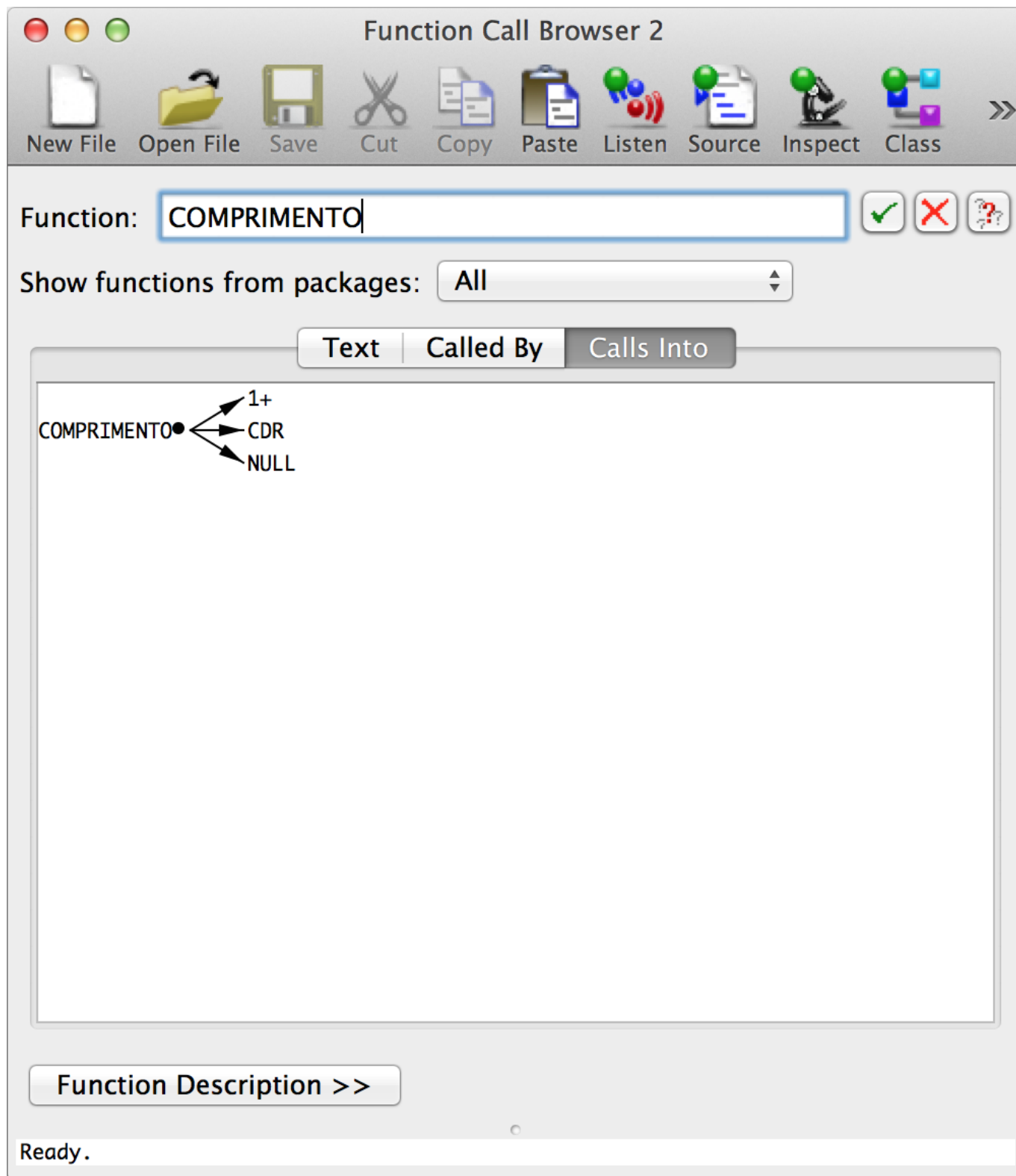


Figura 3: Resumo das funções chamadas pela função `comprimento`.

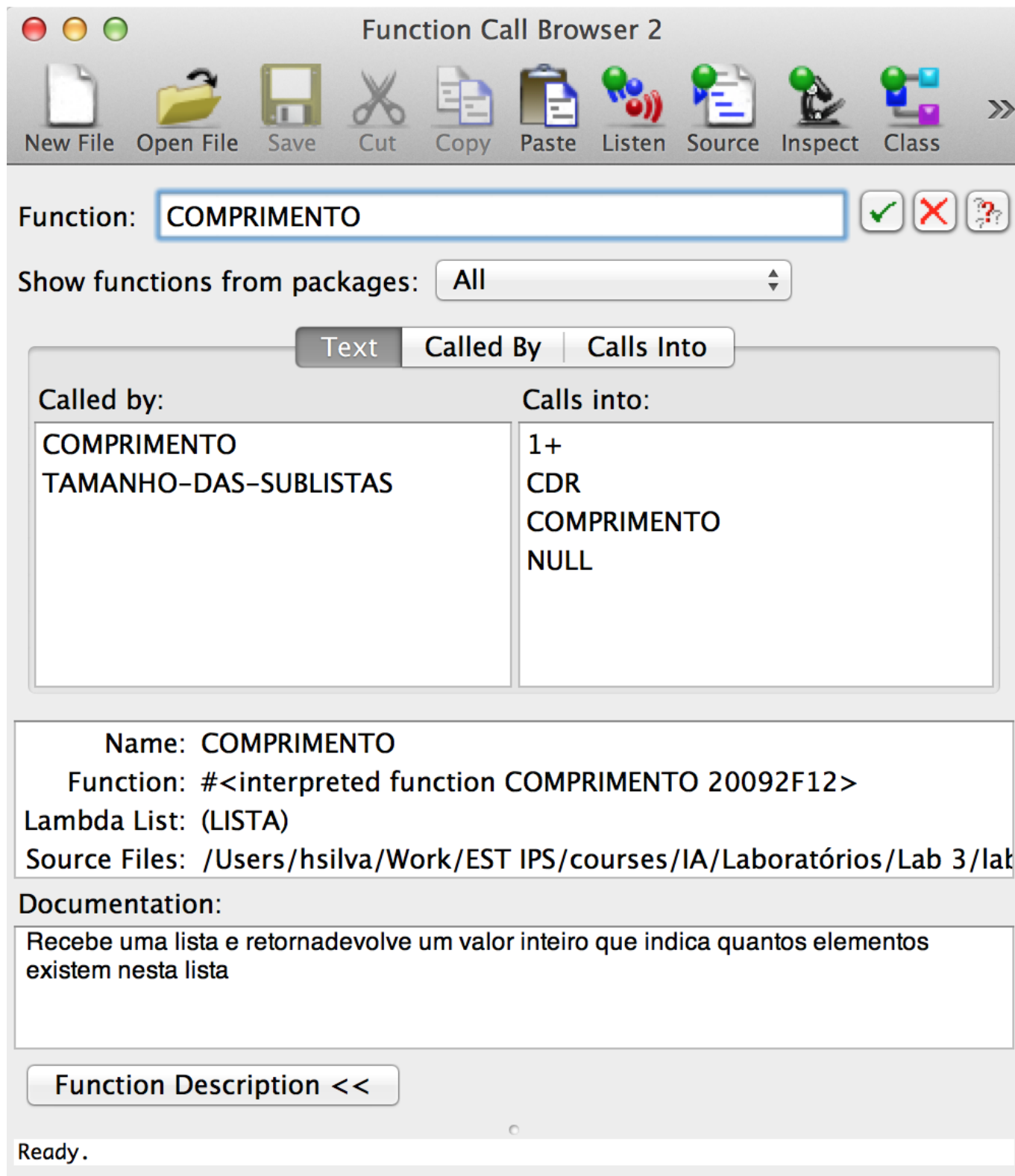


Figura 4: A Janela do *Function Call Browser*.

2.2. Visualizar a Pilha de Chamadas a Funções

A instrução **trace** permite rastrear cada chamada e cada retorno de uma determinada função, qualquer que seja o sítio onde a função se encontra no programa. Para rastrear uma função (fazer o **trace**), pode utilizar o *Listener* ou o módulo específico do LispWorks.

No *Listener*, execute a instrução **trace** com o nome da função. Pode executar o **trace** com mais de um nome de função. Quando o programa executa uma função rastreada, irá imprimir o nome da função no terminal do

interpretador e, na maioria dos IDE Lisp, irá imprimir também os argumentos de entrada com os quais a função foi invocada, bem como os valores retornados na saída da função.

Para deixar de rastrear uma função, deve utilizar a instrução **untrace** e passar-lhe o nome da função. O exemplo seguinte ilustra a utilização da funcionalidade **untrace** no LispWorks.

1. Abrir o módulo *Trace* do LispWorks, clicando no ícone ilustrado na Figura 5.
2. A janela deste módulo aparece tal como representada na Figura 6.
3. Nesta janela do *Tracer*, insira o nome da função **comprimento** no espaço reservado à chamada *Trace* e carregue na tecla ENTER. O nome da função irá ser adicionado na zona *Traced Functions* desta janela.
4. Nas Figuras 7 e 8 poderá ver exemplos dos resultados obtidos através do *Tracer* para chamadas simples à função **comprimento**.
5. No *Listener* insira agora a seguinte chamada à função **tamanho-das-sublistas**.

```
CL-USER > (tamanho-das-sublistas '((1 2 3 4) (5 6 7 8)))
```

6. Na janela do *Tracer*, escolha o separador *Output Data*. Irá verificar que o resultado mostra duas chamadas à função **comprimento**, como esperado. Note que:
 - Os símbolos azuis representam as chamadas à função
 - Os símbolos verdes representam os argumentos de entrada
 - Os símbolos vermelhos representam os resultados
7. Para deixar de observar a função **comprimento**, escolha a opção **untrace** existente na zona *Trace State* da janela do *Tracer*.



Figura 5: O ícone do *Tracer*.

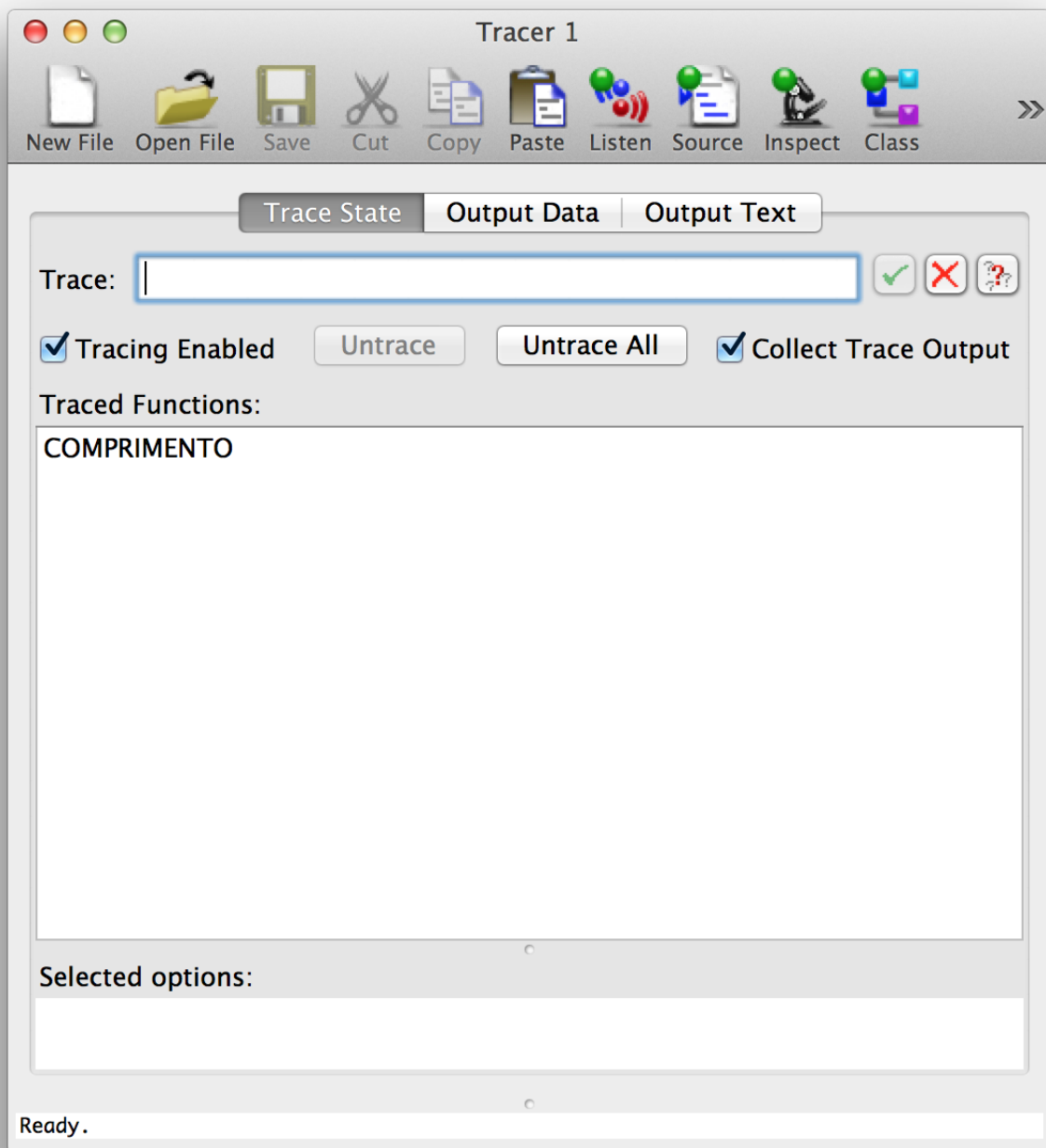


Figura 6: Janela do *Tracer*.

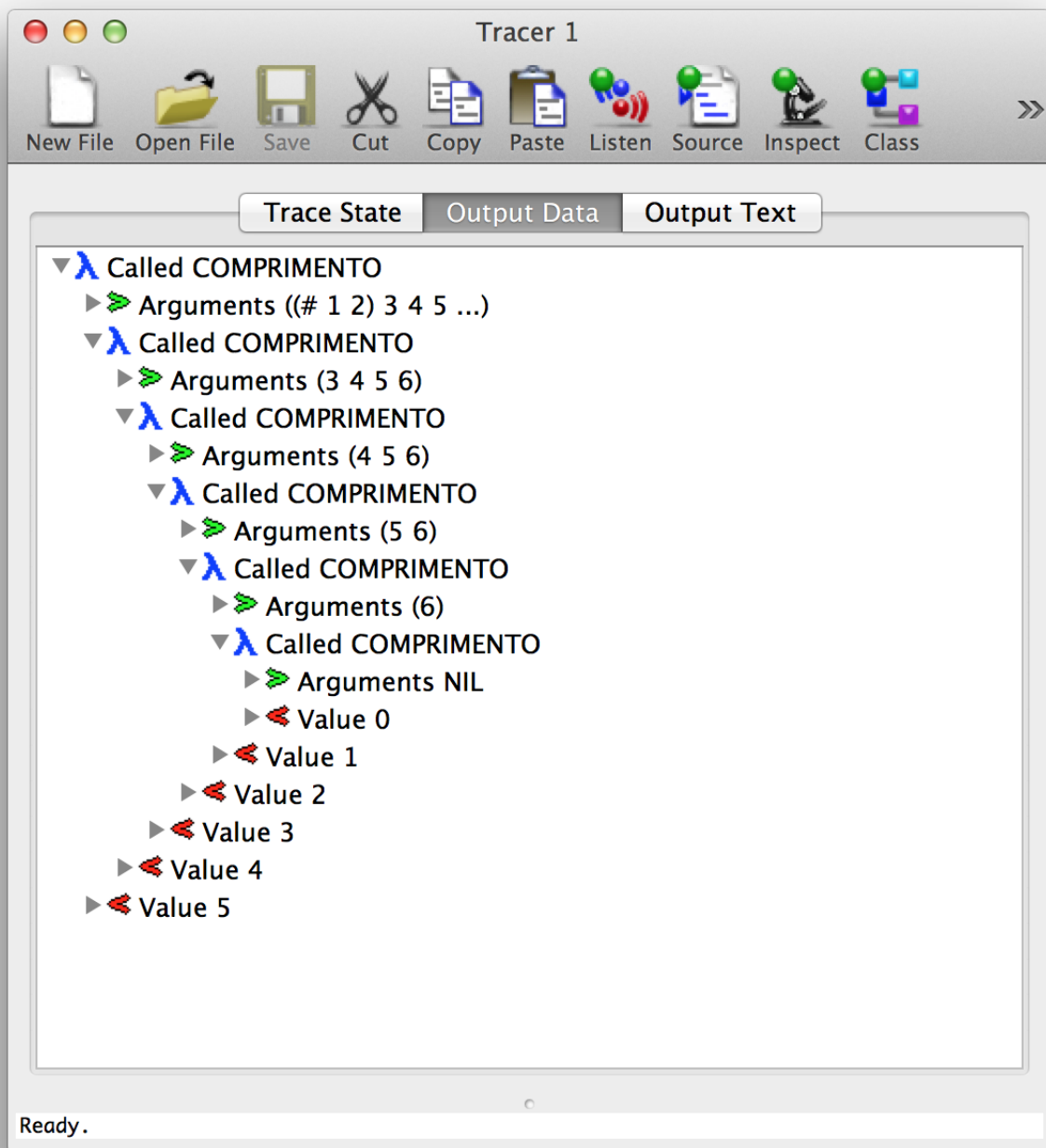


Figura 7: Exemplo da sequência de chamadas à função `comprimento`, tendo como argumento de entrada a lista `((1 2) 3 4 5 6)` apresentada através da interface gráfica.

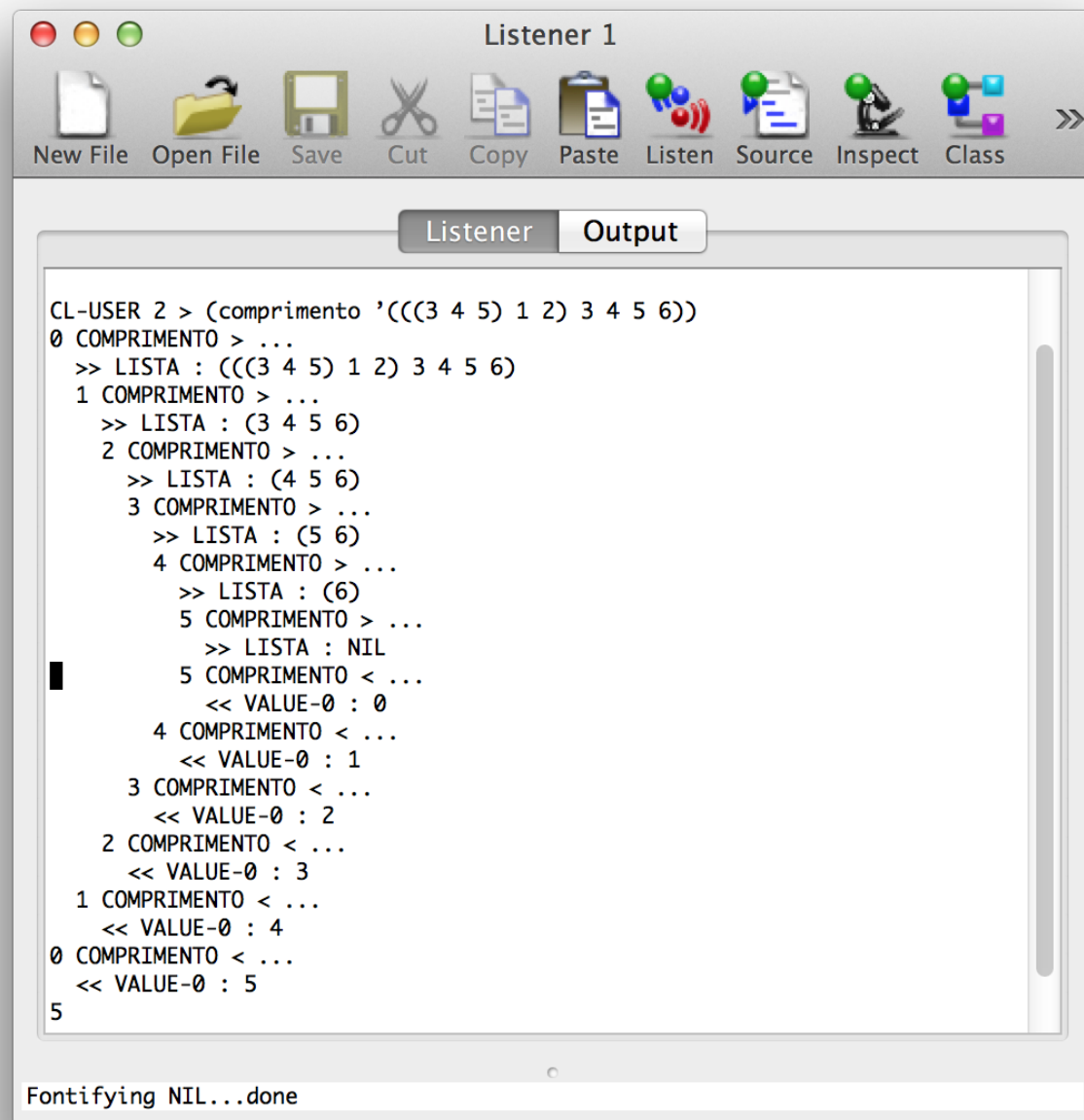


Figura 8: Exemplo da sequência de chamadas à função `comprimento`, tendo como argumento de entrada a lista `((3 4 5) 1 2) 3 4 5 6))` tal como apresentada no terminal do interpretador.

2.3. Sugestões para Apoio ao Desenvolvimento de Funções

- Teste as funções ou até parte delas à medida que as escreve;
- Defina funções curtas e teste-as logo após as ter escrito;
- Escreva casos de teste e inclua-os como comentários no mesmo ficheiro. Por exemplo:

```
;;(comprimento '(1 2 3 4 5)) ;teste a  
COMPRIMENTO tendo uma lista como argumento
```

3. Exercícios sobre Funções Recursivas

Implemente as seguintes funções com base no conceito de recursividade.

1. **factorial**: calcula o factorial de um número passado como argumento de entrada.

```
CL-USER > (factorial 10)
3628800
```

2. **n-esimo**: retorna o n-ésimo elemento de uma lista, ou seja, **n-esimo** replica o comportamento da função Lisp **nth**: `(nth n 1)`

```
CL-USER > (n-esimo 3 '(2 5 10 22 30))
22
```

3. **soma-lista**: calcula o valor correspondente à soma de todos os elementos de uma lista composta por valores numéricos

```
CL-USER > (soma-lista '(1 2 3 3 4 4 5 ))
22
```

4. **existe**: verifica se um elemento procurado, passado como argumento de entrada, existe numa lista passada como argumento de entrada, e caso exista devolve a cauda da lista em que esse elemento é a cabeça; a função **existe** replica o comportamento da função Lisp **member**: `(member o 1)`

```
CL-USER > (existe 3 '(1 2 3 3 4 5))
(3 3 4 5)
```

5. **junta**: retorna a união entre duas listas, ou seja, **junta** replica o comportamento da função Lisp **append**: `(append l1 l2)`

```
CL-USER > (junta '(1 (2 3) ) '(3 (4) 5) )
(1 (2 3) 3 (4) 5)
```

6. **inverte**: retorna a lista passada como argumento de entrada invertida, ou seja, **inverte** replica o comportamento da função Lisp **reverse**: `(reverse 1)`

```
CL-USER > (inverte '(1 2 3 (5 6) 7) )
(7 (5 6) 3 2 1)
```

7. `conta-atomos`: retorna o número de átomos que uma lista contém.

```
CL-USER > (conta-atomos '(1 2 3 (5 6) 7))  
6
```

8. `alisa`: devolve todos os elementos de uma lista que poderá conter sub-listas, com todos os elementos agregados numa única lista principal.

```
CL-USER > (alisa '(1 2 3 (3 4) (4 5) ))  
(1 2 3 3 4 4 5)
```