Ficha de Laboratório Nº 1: Instalação do IDE LispWorks e exercícios de Lisp

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal

Prof. Joaquim Filipe
Eng. Filipe Mariano

Objetivos da Ficha

- Introduzir o IDE LispWorks
- Aprender a utilizar a consola *Listener*
- Definir e testar expressões em *Common Lisp*

1. Instalação do LispWorks

Para instalar a aplicação, deverá registar-se em: http://www.lispworks.com/downloads/index.html

Depois de escolher o sistema operativo adequado, poderá descarregar o ficheiro de instalação.

2. Introdução ao IDE LispWorks

O IDE LispWorks é constituído por várias ferramentas que correspondem a grupos lógicos de funcionalidades associados ao desenvolvimento de aplicações em Lisp. Dentro do conjunto de ferramentas, o **listener**, o **editor** e o **debugger** serão as mais usadas para desenvolver os projetos na unidade curricular de IA.

O *listener* é um terminal fornecido no IDE para avaliar expressões escritas em Common Lisp. O listener é um outro nome dado para o ciclo read-eval-print (REPL), que permite avaliar de forma interativa uma expressão Lisp e obter os seus valores de saída / retorno. Esta ferramenta é fundamental para testar o código sem que seja necessário a compilação ou a avaliação de ficheiros inteiros de código-fonte Common Lisp.

O editor interno do LispWorks permite escrever e desenvolver código em Common Lisp, disponibilizando um conjunto de funções especificamente concebidas para o efeito. Está totalmente integrado no ambiente de desenvolvimento, de forma a que o código possa estar imediatamente disponível para testes. Permite criar e editar ficheiros com código em Lisp.

O *debugger* é a ferramenta que permite analisar a execução do código fonte e depurar eventuais problemas existentes no mesmo.

O browser de *outputs* permite recolher e examinar os resultados da execução de expressões / programas, nomeadamente mensagens de aviso e de erro apresentadas durante a compilação e a saída gerada pela execução de funções. Outras ferramentas do IDE também podem ser usadas para observar os resultados, o que permite ao programador ter acesso a qualquer saída em função da ferramenta em uso.

O *inspector* permite examinar e modificar o conteúdo de objetos Common Lisp. É uma ferramenta útil durante o desenvolvimento, uma vez que permite anlisar o estado de variáveis, em qualquer altura, durante a execução. Assim, é fácil ver o valor de um slot e, se necessário, alterar o seu valor, de modo a que possa testar os efeitos de tal alteração antes de fazer as mudanças necessárias no próprio código fonte.

O *object clipboard* (ou área de transferência) é usado para gerir objetos Lisp. É possível selecionar qualquer objeto na área do clipboard para usar em operações do ambiente de trabalho.

3. Utilização do LispWorks

3.1. Iniciar o LispWorks

Para iniciar o LispWorks no Microsoft Windows:

- Clicar em **Iniciar** na barra de tarefas e na caixa de pesquisa escrever **LispWorks**
  Ou
- No menu **Iniciar**, selecione **Programas > LispWorks X.Y Personal**

Aparece uma janela de abertura, seguida da janela principal do IDE (chamada *podium*), detalhada na Figura 1. A janela do *Listener* também aparecerá se a sua imagem estiver configurada para iniciar logo no arranque (o que é o caso por defeito).

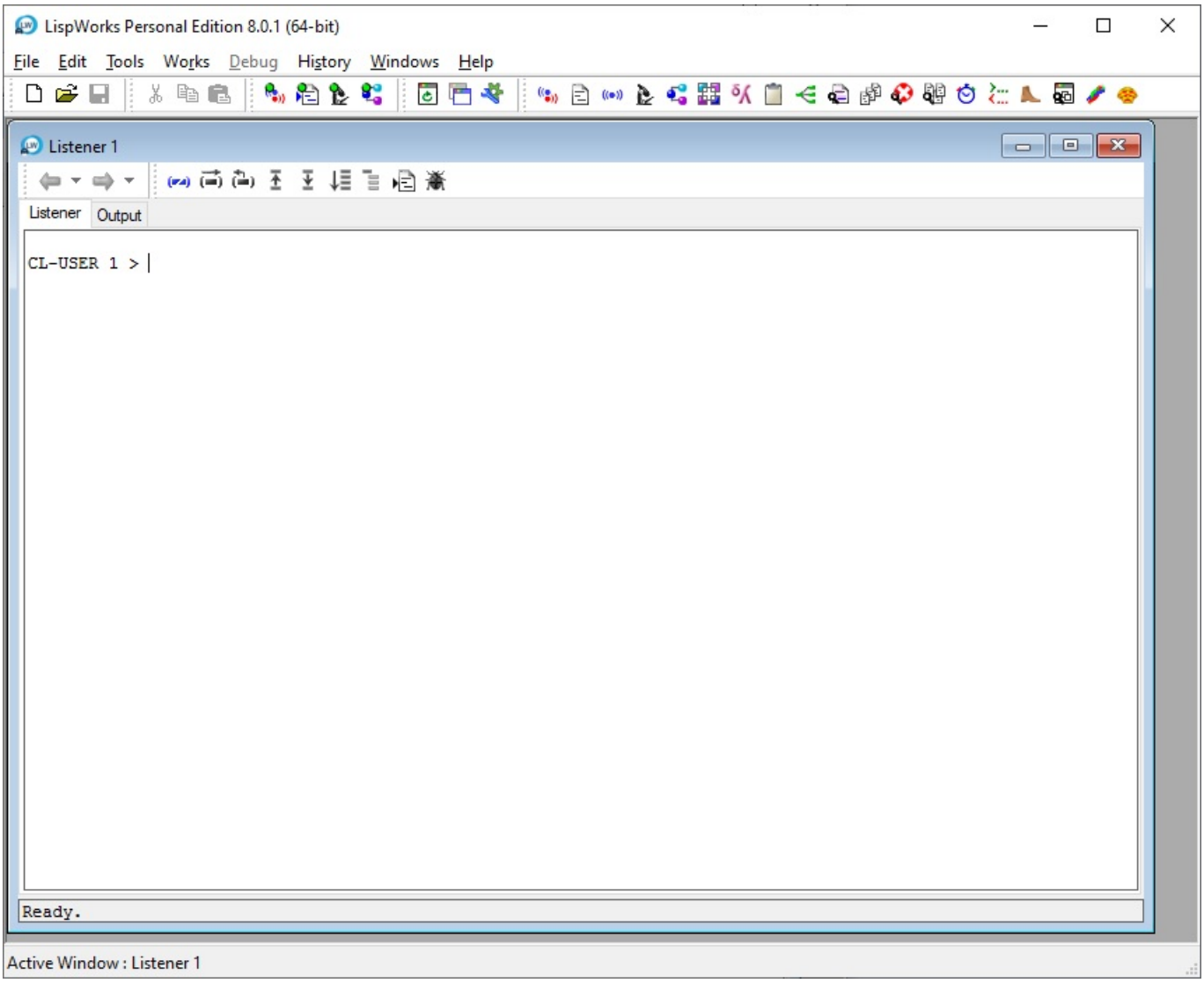

Figura 1: A janela principal do IDE LispWorks

A janela principal é mostrada automaticamente sempre que iniciar o LispWorks. Contém menus, barra de ferramentas (sob a forma de ícones) e uma área com indicação da janela ativa. A barra de ferramentas fornece um acesso rápido a alguns dos comandos mais usados. Como muitas outras aplicações, a barra de menu opções de acesso a **Ficheiros** (Files), **Edição** (Edit), **Ferramentas** (Tools), **Gestão de Janelas** (Windows), **Ajuda** (Help), e um menu específico LispWorks chamado (**Works**).

O menu **Works** contém comandos que se aplicam à janela ativa do ambiente de trabalho. O título da janela ativa é mostrada na janela principal do LispWorks, debaixo da barra de ferramentas. O menu **File** permite abrir um ficheiro num editor, ou imprimir um ficheiro, independentemente de qual é a janela ativa. Quando as ferramentas **Editor** ou **Listener** estão ativas, o menu **File** contém outros comandos para operações possíveis de realizar sobre o ficheiro activo.

O menu **Tools** dá acesso à todas as ferramentas do IDE LispWorks. O menu **Windows** mostra a lista de todas as janelas ativas no ambiente de trabalho que estejam em execução.

> **Nota:** Para sair do LispWorks, escolha **File > Exit**. Também existem menus contextuais que variam em função do tipo de janela ativa. Quando o *Listener* está seleccionado, é possível aceder aos menus **Values** e **Debug**. Quando o editor está selecionado, os menus **Buffers** e **Definitions** são ativados.

3.2. Utilizar o *Listener*

Os programadores de Lisp costumam desenvolver de forma incremental e testar no listener antes de guardar o código de trabalho para o disco. Durante uma sessão típica de desenvolvimento, o programador avalia pedaços de código no *Listener*, examina os efeitos utilizando as outras ferramentas do IDE, e volta para o *Listener* sempre que quer avaliar outro pedaço de código. Se a janela do *Listener* ainda não apareceu (verifique o menu Windows), inicia uma nova, escolhendo **Tools > Listener** ou clicando no ícone da Figura 3 na barra de ferramentas da janela principal.

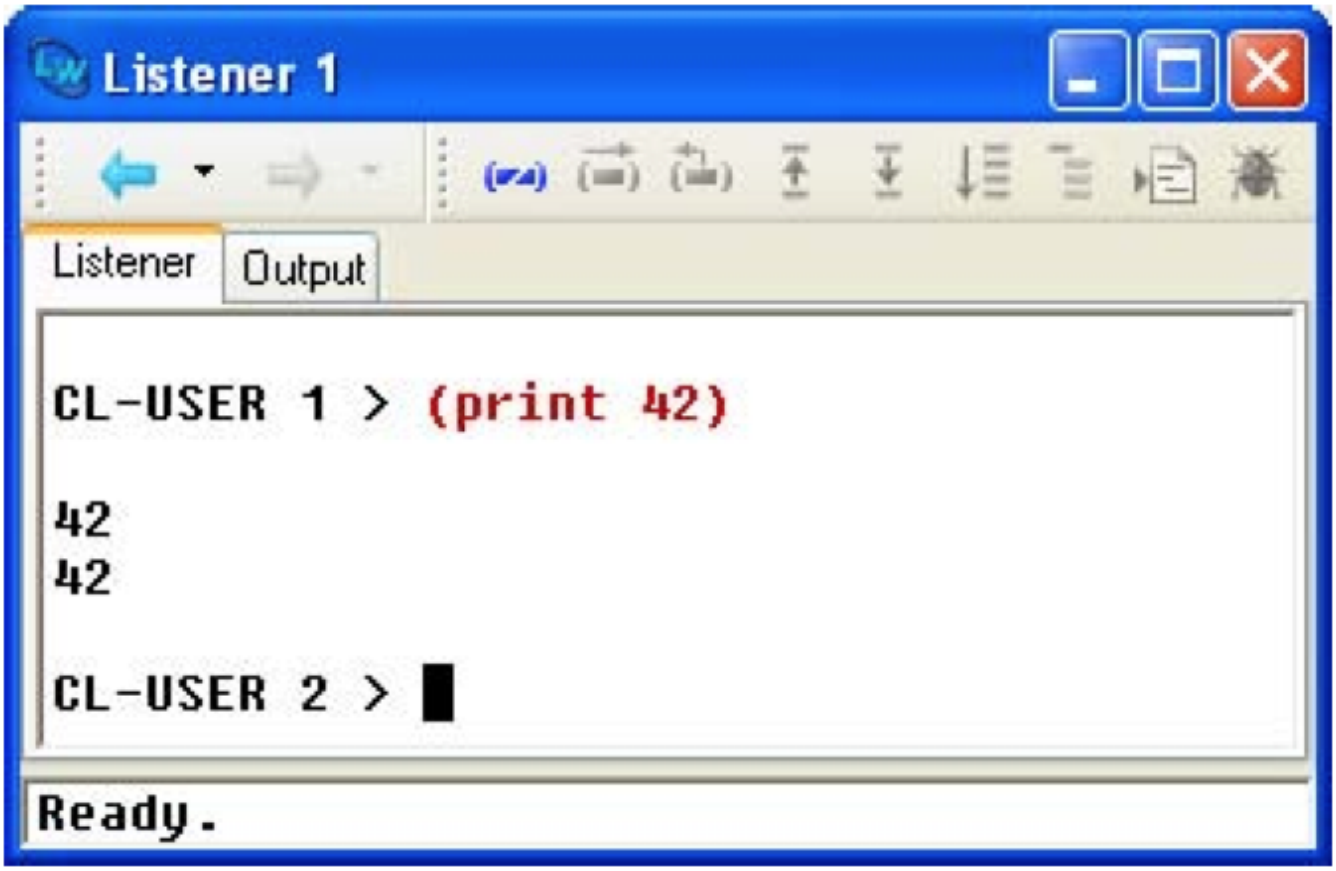

Figura 2: A janela do *Listener*

Figura 3: O ícone do *Listener* na barra de ferramentas

O *Listener* contém dois pontos de vista: o ponto de vista *Listener* e a vista de saída (output). É possível mudar de um ponto para o outro, clicando nas guias **Listener** e **Output**, respectivamente. É possível avaliar expressões em Lisp no ponto de vista *Listener*, digitando a expressão, seguida da tecla Enter. Qualquer saída produzida é exibida na consola do *Listener*.

O *Listener*, por defeito, aparece com uma linha de comando (chamado *prompt*), no início da janela. As expressões Lisp por avaliar são inseridas nesta linha, a seguir ao prefixo.

> **Nota:** A linha de comando ou *prompt* apresenta-se sob a forma de uma linha com prefixo sequencial, tal como "CL- USER 1>", composta pelo nome do pacote atual (isto é, o valor de cl: pacote), seguido por um número inteiro positivo

1. Digite a expressão (+ 1 2) no *Listener* e pressione **Enter**.

```
CL-USER 1 > (+ 1 2)
3
CL-USER 2 >
```

   O resultado da avaliação, 3, aparece no *Listener*, e é apresentada uma nova linha. Observe que o número da linha foi incrementado, indicando que uma expressão foi avaliada. A numeração é fornecida para ajudar o programador, dado que pode vir a inserir várias expressões semelhantes.

> **Nota:** Se aparecer uma mensagem de erro, simplesmente aborte e tente novamente. Se não for apresentado nenhum resultado, certifique-se de que a expressão digitada contém parênteses de fecho e de abertura. Se não for o caso, o *Listener* não irá avaliar até que seja inserida uma expressão Lisp completa.

2. Coloque o cursor do rato no fim da linha (1) que acabou de avaliar e Prima **Enter**. A expressão que acabou de ser avaliada é impressa numa nova linha. Se pressionar **Enter** irá avaliar esta expressão novamente, ou, mais útil, pode editar a expressão ligeiramente antes da avaliação.

3.3. Utilizar o *Object Clipboard*

Como exemplo de adição de um objeto Lisp para a área de transferência de objetos, siga estes passos:

1. Avaliar uma expressão Lisp na janela *Listener*. Seu valor é impresso.
2. Escolha o comando do menu **Works > Values > Clip**.

O valor vindo do *Listener* está agora na área de transferência de objetos. Se o *object clipboard* ainda não estiver visível, então, de seguida, opte pelo comando **Tools > Object Clipboard**.

3.4. Problemas e Sugestões

Problema 1: Quando se copia parte de um texto ou instruções de um ficheiro em formato PDF para o *Listener* ou para o editor do LispWorks, podem acontecer algumas situações para as quais devem estar prevenidos. Acontece frequentemente que sejam copiados caracteres especiais que não são visíveis ou que algum dos caracteres copiados percam a formatação. Se isso acontecer, o texto copiado pode introduzir erros e gerar um comportamento errático do LispWorks.

Solução 1: Se o listener não se comportar da forma esperada, o utilizador deverá fechar esta janela e abrir uma nova janela, utilizando o ícone apresentado na Figura 3.

4. Exercícios

Tente avaliar individualmente as expressões seguintes, no papel, e confirme os seus resultados recorrendo ao *Listener*:

4.1. Avaliação de Expressões

Qual é o resultado das seguintes expressões?

1. ```
   (- (+ 3 (* 7 6) 5 (/ 4 2)) 30 22)
   ```

2. ```
   (- (/ (/ 20 2) (/ 15 3)) 1)
   ```

4.2. Operações Aritméticas

Crie expressões que correspondem às opererações aritméticas seguintes e teste no *Listener* as expressões que definiu:

1. ```
   (3+5+6+2)-1
   ```
2. ```
   (3+5+6)+2-1
   ```
3. ```
   3+5+6/(2-1)
   ```
4. ```
   (8+2)*5
   ```
5. ```
   (29+11-(2-4))/(6*7)
   ```

4.3. Selectores

Qual é a cabeça das seguintes listas?

1. ```
   (2 -2 4)
   ```
2. ```
   ((3) 4 5)
   ```
3. ```
   (((3) 0 (4 5)) ())
   ```
4. ```
   (nil)
   ```

Qual é o resto das mesmas listas?
   ```

4.4. Predicados

Qual é o resultado das seguintes expressões?

1. ```
   (atom 3)
   ```
2. ```
   (atom '(/4 2))
   ```
3. ```
   (atom (/ 4 2))
   ```
4. ```
   (listp '( 3 4))
   ```
5. ```
   (listp 3)
   ```
6. ```
   (listp '3)
   ```

4.5. Acesso a Listas

Diga qual a expressão contendo `car`'s e `cdr`'s que permite obter o átomo `5` a partir das seguintes listas

1. ```
   (3 4 5 6)
   ```
2. ```
   (3 4 (1 3 (6 5)) (1 2))
   ```

Qual é a diferença entre a avaliação de:

1. ```
   (car (car '((1) 2))) e de (car '(car ((1) 2)))
   ```

   e
   ```

2. `(cons (+ 1 2) '(3))` e de `(cons '(+ 1 2) '(3))`

?

4.6. Construção de Listas

Diga qual o resultado das seguintes expressões:

1. `(cons 1 nil)`
2. `(cons 3 '(5))`
3. `(cons '(3) '(2))`
4. `(cons '(3) ())`
5. `(cons '(3 (5 6)) '((2 4) 6))`

Diga qual a expressão simbólica contendo o número máximo de `cons` que permite construir as seguintes listas:

1. `(3 (4) 5)`
2. `(3 ((4) 6) 5)`

5. LispWorks *Editor*

5.1. Funcionamento Geral

Enquanto que o *Listener* é prático para escrever e testar expressões curtas de Lisp, o LispWorks dispõe de um editor que permite escrever, carregar e armazenar código Lisp extenso de forma mais cómoda.

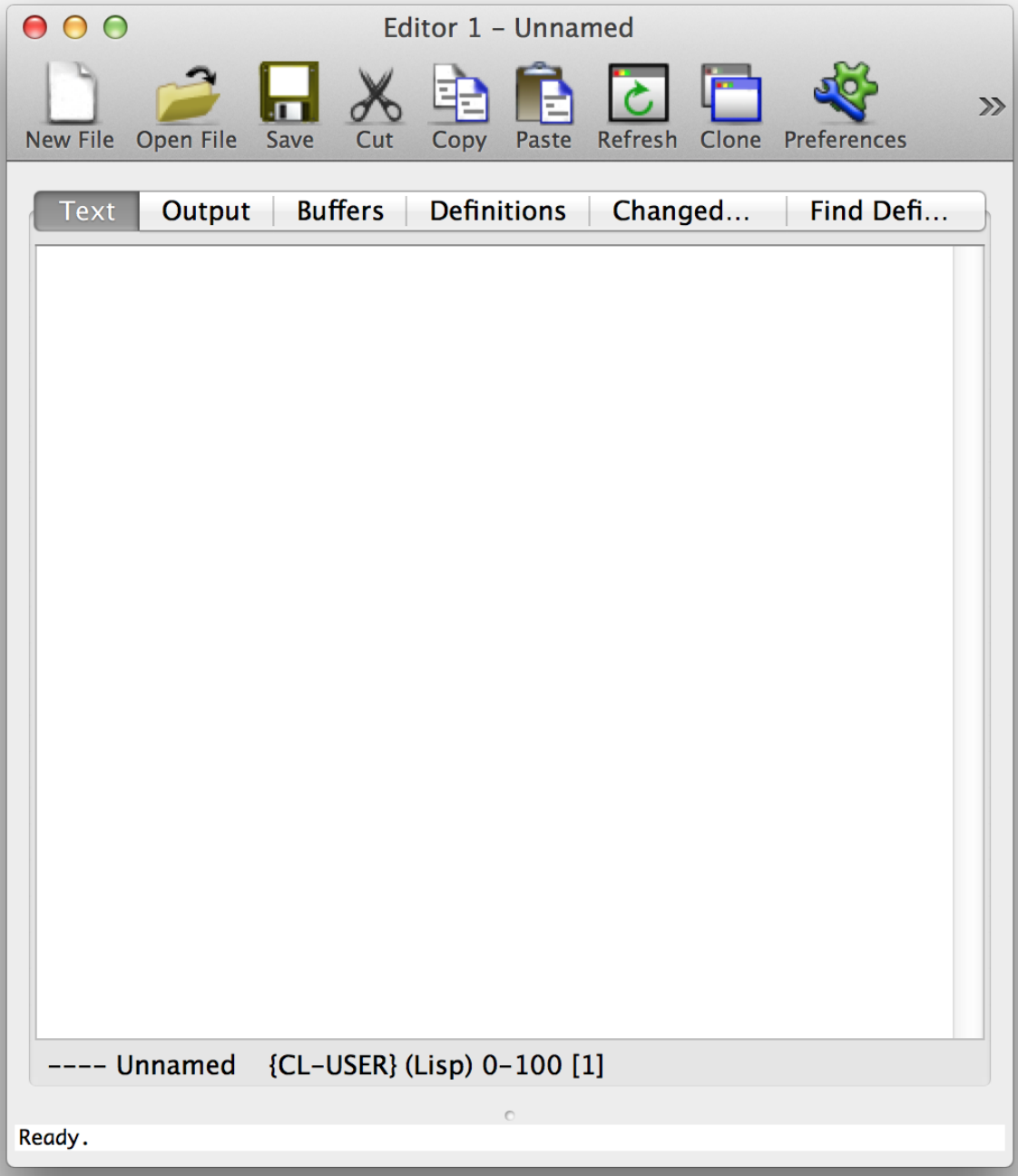

Figura 4: Janela de edição

1. Para abrir o editor, escolha as opções **File > New** ou **Window > Tools > Editor**.
2. Na vista *Text* poderá escrever o seu código Lisp, beneficiando de auxiliares visuais úteis como destaque da sintaxe, confirmação da abertura e fecho de blocos, entre outros
3. Pode guardar os seus ficheiros *.lisp* através das opções **File > Save** ou **File > Save as...** e ler os mesmos através da opção **File > Open...**
4. Para executar o código Lisp existente no editor, deverá utilizar a opção **Buffer > Evaluate**. Os resultados podem ser observados na vista *Output*.

5. Para compilar o código Lisp existente no editor, deverá utilizar a opção **Buffer > Compile** (ou o atalho **Compile Bufer**) para compilar todo o ficheiro, ainda que possa utilizar apenas a opção **Compile** para compilar linha actual.

   > **Nota:** A compilação irá comutar automaticamente a vista para *Output*, de modo a exibir os resultados de compilação. Pressionando a tecla espaço a vista será automaticamente revertida para o modo *Text*.

6. As funções compiladas tipicamente aceleram a execução mais de dez vezes do que a interpretação do código fonte. É possível gerar os binários optimizados para um dado ficheiro contendo código fonte, a partir do *Editor*, utilizando da opção **File > Compile and Load**. Como resultado desta operação será guardado um ficheiro com extensão .**asl* na pasta de trabalho.

   > **Nota:** Os ficheiros .*fasl* podem ser carregados no *Listener* através da opção **File > Load...**, ficando as funções neles contidas disponíveis para utilização.

5.2. Exercício

Para testar as funcionalidades descritas na secção anterior, realize as seguintes experiências:

1. No *Listener* execute o operador abaixo:

   ```
   (quadrado 2)
   ```

   O resultado será uma mensagem de erro a indicar que o operador `quadrado` não está definido.

2. Crie um novo ficheiro de código fonte e defina o operador `quadrado` da seguinte forma:

   ```
   (defun quadrado(n)
     (* n n))
   ```

3. Guarde o ficheiro com um nome e numa pasta à sua escolha

4. Produza o ficheiro `lab1.lisp` com a função `quadrado`

5. Reinicie o LispWorks

6. Experimente repetir o Passo 1
   O resultado será uma mensagem de erro a indicar que o operador `quadrado` não está definido.

7. Carregue o ficheiro `lab1.lisp`, compile e repita o Passo 1
   Desta vez o resultado será o cálculo efectuado pelo operador.

6. Documentação

Como boa prática de programação, é importante documentar todo o código para explicar o seu funcionamento. É um recurso útil tanto para o próprio programador salvaguardar os casos em que terá de revisitar o códido mais tarde, como para facilitar a manutenção e/ou actualização futura por parte de terceiros.

6.1. Comentários

Em Lisp é possível comentar blocos de código, colocando os conteúdos entre a sequência `#|  |#` (ex: `#| <BLOCO COMENTADO> |#`). Os comentários em linha podem ser feitos com o caracter ponto e vírgula (`;`) e, em geral, devem respeitar as seguintes regras:

- Utilizar um ponto e vírgula, nos casos em que está a adicionar informação de contexto a uma linha de código. O comentário deverá aparecer no fim da linha, separado do código fonte com um espaço e idealmente identado de forma a se alinhar verticalmente com outros comentários colocados na mesma região.
- Utilizar dois pontos e vírgula para comentários colocados entre linhas de um grupo de expressões.
- Utilizar três pontos e vírgula para comentários que se aplicam ao nível superior de uma expressão ou grupo de expressões.
- Utilizar quatro pontos e vírgula para comentários que se aplicam a grandes secções num ficheiro de código fonte (ex: cabeçalhos).

6.2. docstring

Excepto nos casos em que as função são auto-explicativas, é importante complementar as funções desenvolvidas com strings de documentação (*docstrings*). Em Lisp estas strings são blocos de texto colocados entre aspas (ex: `"<DOCSTRING>"`).

Estas podem ser associadas a funções, tipos, variáveis ou outros elementos, e geralmente são lidas por IDEs ou interrogações REPL, além de possibilitarem a geração automática de documentação por conversão, por exemplo, para páginas web.

Idealmente os comentários descrevem como usar o código (incluindo casos de uso a evitar para não gerar erros), por oposição a explicar como funciona o código (dado que esta parte fica a cargo dos comentários).

No caso das funções, as *docstrings* devem descrever a estrutura das mesmas, nomeadamente, o que a função faz, o que significam os argumentos, quais são os valores de retorno e que condições pode a função abordar. Os símbolos de Lisp (como por exemplo os argumentos das funções) devem ser escritos em maiúsculas. Um exemplo de *docstring* é:

```
(defun divisao (n d)
  "O valor de D deve ser diferente de 0. Devolve N/D."
  (/ n d))
```

Pode visualizar a *docstring* definida para a função através da seguinte instrução no listener:

```
(documentation 'divisao 'function)
```