Lisp – introductory tutorial

objectives

1. familiarisation with the Lisp environment
2. experimentation with association lists
3. initial experimentation with Lisp forms

The work here will introduce you to the Lisp IDE and give you some initial exposure to the style of code development you use for languages like Lisp. You will use a few predefined Lip forms/functions to develop some functions of your own. Like all other tutorials, you will need to consult lecture notes to complete this.

the way to work

pair-programming – outside of tutorial time you may work on your own, during tutorials you will work in small groups (preferably two people, maximum three).

the Lisp IDE – the Lisp editor has a number of Lisp-specific features which will help you produce code, these features impose little learning burden on you (they take place in the background) but will become important as tutorials progress. It is essential that you use the editor within the IDE. Staff will **not** help you with your code if it's in Notepad, winedit, etc.

timing/effort – you are timetabled for 1hour a week in a lab. The tutorials will take you longer than this to complete so you will need to do additional work outside of tutorial time. The first tutorial is the only one which is short enough to be completed in one week, others will last longer. You will make best use of tutorials (and the staff) if you come prepared with things you would like to discuss or that you need help with.

design of tutorials – tutorial work is split into a series of tasks which you should complete in sequence. The aim of these tasks is not for you to produce a product but rather for you to investigate some area of practical and/or theory from your course. Typically you will need to do some experimentation and/or go back over lecture notes to complete the work.

getting started

please do the following...

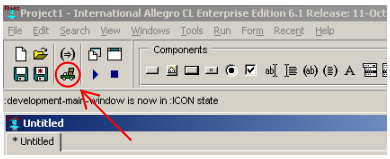
1. set up a directory/folder for the lisp rsources you will download for this module. These notes assume this folder will be "**U:\lisp**"
2. go to www.agent-domain.org and download the **matcher** from the downloads section, put this in **U:\lisp**
3. downloads "**utils.lsp**" from \resources\Lisp on the site – put this in **U:\lisp** as well.
4. create a new text file called **startup.cl** in your **U:\lisp** folder and cut & paste the following lines of code into it...

```
(load "U:\\lisp \\matcher(1.6).lsp")
(load "U:\\lisp \\utils.lsp")
```

starting lisp

you need to complete the following steps each time you start the Lisp environment...

- find the allegro program icon & open the Lisp IDE (note use allegro.exe **not** allegro-ansi.exe)
- the load & compile button (see diagram below) opens a file selection window, use this to select and load the your **U:\lisp\startup.cl** file then type **(use-package :matcher)** into the debug window



You may have as many as 4 windows open within the Allegro IDE...

1. a form/GUI editor – we will not be developing GUIs so you can close this
2. a smallish window titled "inspect", this shows a table of system symbols and their values – you can close this too (we will open inspectors as we need them)
3. a window titled "debug" – you need this one, it is your main interaction & experimentation window
4. another one (possibly) which has the label "untitled" – this is an edit window, keep this open if you have it, don't worry about it if it is not there when you open the IDE.

If you are starting this tutorial before you have had an introductory session on Lisp please complete task 0 otherwise continue to task 1.

task 0

From the lecture series in Symbolic Computation at www.agent-domain.org follow through the seminar trace from the 1st lecture. Experiment with the example Lisp expressions in your Allegro debug window. Part way through this exercise you will need to load the Lisp definitions provided. Cut and paste these into your Allegro editor window and compile them. Remember: you can compile a definition from the editor window by placing your cursor inside the definition & clicking the button marked **(⇒)** on the Allegro task bar.

The first few tasks will build a small suite of expressions to translate words representing single digit numbers from one language to another, something like...

```
(translate 'two 'French) → deux
```

task 1

Build an association list (ASL) called *numbers* to associate the numbers 1-5 with the words used to spell them in three different languages other than English (ask around the room if you don't have the language knowledge for this). Use **defparameter** & type this in to the editor window. Organise the ASL with the language name as the primary association and the number (in digit form) as the secondary association. So, for example...

```
(-> *numbers* 'Hindi 1) → ek
```

Remember: compile a definition from the editor window by placing your cursor inside the definition & clicking the button marked **(⇒)** on the Allegro task bar.

task 2

write a flat (single level) ASL to associate numbers spelt in English with their numeric form so...

```
(-> *english-numbers* 'two)  → two
```

task 3

write a function called translate which uses the ASLs above to translate from numbers spelt in English to one of your other languages...

```
(translate 'two 'French) → deux
```

task 4

write 3 short functions to translate to each of your languages, so for example...

```
(in-french 'two) → deux
```

Remember: you can compile multiple definitions from the editor window by highlighting them all & clicking the button marked **(⇒)**

task 5

write a function (which uses **mapcar**) which takes a translation function and a list of English numbers & performs multiple number translation...

```
(translate-v2 #'in-french '(five three four))
→ (cinq trois quatre)
```