Continue

SWI-Prolog syntax is based on the Edinburgh Prolog syntax, which is similar to the ISO-Prolog standard syntax. SWI-Prolog offers several extensions beyond the ISO standard, including support for nesting /* ... */ comments and an extended character escape syntax within quoted atoms. The processor character set specifies the class of each character used for parsing Prolog source text, with Unicode being fixed as the classification system. The syntax allows for various special characters to be represented using escape sequences, such as \a for the alert character and \e for the escape character. The list of escape sequences is compatible with the ISO standard but also contains some extensions. In ISO mode, only certain escape sequences are skipped, while in native mode, white space that follows the newline is also skipped, resulting in a deprecation warning. It's recommended to use \c or put the layout before the \ for compatibility and future-proofing. The specification of a character can be done in various ways, including using the backslash (\) followed by a hexadecimal code, such as \xa\3, which emits the character 10 (hexadecimal `a') followed by `3'. This notation is used for Unicode characters. Alternatively, the \uXXXX notation can be used to specify a Unicode character using exactly 4 hexadecimal digits, which is an extension to the ISO standard. The \UXXXXXXXX notation is similar, but uses 8 digits to cover the entire Unicode set. Octal character specification is also possible using the \40 notation. The backslash itself can be escaped using \\, and other characters such as single quote (\''), double quote (\"), and back quote (\`) can also be specified. Character escaping is only available if the current_prolog_flag(character_escapes, true) is active. SWI-Prolog also supports non-decimal numbers, including binary, octal, and hexadecimal numbers, which can be written in either Edinburgh or ISO syntax. For example, 0b100 \/ 0xf00 is a valid expression. Large integers can be split into digit groups using the underscore (_) or space characters. The ~I format specifier can be used to print integers with this notation. Additionally, SWI-Prolog supports reading and printing special floating point values, including Infinity and NaN (Not a Number), which can be printed as 1.0Inf or -1.0Inf and 1.xxxNaN, respectively. These numbers can be read and produced using the nan/0 function. The arithmetic in SWI-Prolog is compliant with the ISO standard. To accommodate languages that represent the full range of IEEE doubles, SWI-Prolog allows for the creation, reading, and writing of special values such as Inf and NaN. However, it raises exceptions in most cases instead of returning a value. In order to facilitate compatibility with other systems and languages, Prolog by BIM initially introduced an alternative syntax where only the underscore (_) is used to introduce a variable. This syntax is now supported by SWI-Prolog 7.3.27, controlled by the var_prefix flag. Using only underscores to introduce variables improves readability, especially when working with external languages that are case sensitive. For example, in RDF library projects, property and class names often start with uppercase characters. SWI-Prolog also supports Unicode in source files, which extends the Prolog syntax beyond ASCII characters. The Unicode character classification is based on version 6.0.0 of the Unicode standard. Quoted atoms and strings can use any script, while escape sequences \uXXXX and \UXXXXXXXX are used to specify Unicode code points. In terms of identifiers, SWI-Prolog handles them as a single token if they start with ID_Start followed by an ID_Continue sequence. Variables must start with an uppercase character or underscore (_), while atoms do not require any prefix. Control and unassigned characters produce a syntax error outside of quoted atoms/strings and comments. Singleton variables in Prolog are a special type of variable that appears only once in a clause, and can be replaced with an underscore to indicate they should not be modified. These variables can give warnings if used alone, but can also be given names using underscores or uppercase letters to distinguish them from other variables. The system warns about singleton variables appearing more than once, and provides options to suppress these warnings. This depends largely on the OS. You can use pwd/0 and cd/0 to find and change the working directory. The utility ls/0 lists the contents of the working directory. ?- pwd % /home/janw/src/swipl-devel/linux/ true. ?- cd('~/tmp'). true. ?- pwd % /home/janw/tmp/ true. The file likes.pl is installed in a subdirectory demo inside SWI-Prolog's installation directory and can be loaded regardless of the working directory using the command below. See absolute_file_name/3 and file_search_path/2 for details on how SWI-Prolog specifies file locations. ?- [swi(demo/likes)]. true. After this point, Unix users should continue to section 2.1.2. Windows users can start at section 2.1.1.2. 2.1.1.2 Starting SWI-Prolog on Windows When SWI-Prolog is installed on a Windows system, the following important new things are available: A folder (called directory) called swipl containing the executables, libraries, etc., of the system. No files are installed outside this directory. A program swipl-win.exe, providing a window for interaction with Prolog. The program swipl.exe is a version of SWI-Prolog that runs in a console window. The file extension .pl is associated with the program swipl-win.exe. Opening a .pl file will cause swipl-win.exe to start, change directory to the directory in which the file resides, and load this file. 2.1.2 Adding rules from the console Although it's recommended to put your program in a file, optionally edit it and use make/0 to reload it (see section 2.1.4), you can manage facts and rules from the terminal. The most convenient way to add a few clauses is by consulting the pseudo file user. The input is ended using the system end-of-file character. ?- [user]. |: hello :- format('Hello world~n'). |: ^D true. ?- hello. Hello world true. The predicates assertz/1 and retract/1 are alternatives to add and remove rules and facts. 2.1.3 Executing a query After loading a program, one can ask Prolog queries about the program. The query below asks Prolog what food 'sam' likes. The system responds with X = if it can prove the goal for a certain X. ?- likes(sam, X). X = dahl ; X = tandoori ; ... X = chips. 2.1.4 Examining and modifying your program If properly configured, the predicate edit/1 starts the built-in or user-configured editor on the argument. The argument can be anything that can be linked to a location: a file (predicate name, module name, etc.). If argument resolves only one location, editor starts on this location; otherwise, user presented choice. If graphical interface available, editor creates new window, system prompts next command. User may edit source file, save and run make/0 to update modified source files. Editor opens in same console if not in window, leaving editor runs make/0 to reload modified source files. ?- edit(likes). true. ?- make. % /home/jan/src/pl-devel/linux/likes compiled 0.00 sec, 0 clauses ?- likes(sam, X). ... Program can be decompiled using listing/1 as below. Listing argument is predicate name or head (e.g., ?- listing(mild/1), listing matching clauses); without arguments, predicate listing/0 lists entire program.9This lists several hook predicates defined by default; not very informative.? - listing(mild). mild(dahl). mild(tandoori). mild(kurma). true.2.1.5 Stopping Prolog Interactive toplevel stopped with system end-of-file character (Control-D) or halt/0 predicate: ?- halt.$ $

Ai lab manual in prolog. Swi prolog reference manual. Prolog manual pdf. Prolog lab manual pdf. Lemond prolog manual. Kaco powador prolog manual. Manual de prolog en español pdf. Gnu prolog manual. Sicstus prolog manual. Turbo prolog manual pdf. Swi prolog manual pdf. Prolog reference manual. Prolog lab manual r22.