**Introduction to Prolog**

Prolog is a unique and powerful programming language that is based on formal logic and is widely used in fields such as artificial intelligence, knowledge representation, natural language processing, and constraint logic programming. In this report, we will explore the main features of Prolog that make it distinctive and well-suited for these domains.

**Some main features’s Prolog**

**1.1 Declarative Nature:**

One of the key features of Prolog is its declarative programming style, where programmers express relationships and rules as facts and rules. This allows for concise and expressive representation of complex relationships and rules, making it easy to model real-world problems in a natural and intuitive way. For example, consider the following Prolog code that represents the family relationships:

A picture containing text

Description automatically generated

In this example, the facts **father** and **mother** represent the relationships between individuals and their children. This declarative style makes it easy to express complex relationships and rules in a concise and intuitive manner.

**1.2 Prolog's Predicate Notation for Logic.**

Prolog employs predicate calculus, a formal mathematical notation for expressing logical relationships, as its fundamental building block. In Prolog, a program is composed of a collection of predicates, which are used to define relationships, facts, and rules. Predicates are represented as logical statements, with terms and variables that can be unified and matched against other predicates. Prolog uses logical inference and pattern matching to evaluate queries against these predicates, making it a powerful tool for solving complex problems that require reasoning and logical deduction. For example, the following Prolog code defines a rule using predicate calculus notation:

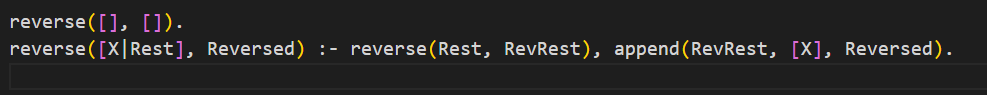
**Graphical user interface

Description automatically generated**

This code states that X is mortal if X is human, using the predicate "mortal" and the ":-" (implies) operator.

**1.3 List Handling and Recursion**

Prolog is a natural language for handling lists and recursion. Lists are fundamental data structures in Prolog, and Prolog provides built-in support for manipulating lists efficiently. Prolog allows developers to easily append, concatenate, reverse, and manipulate lists using predefined predicates. Prolog also naturally supports recursion, which is a powerful technique for solving problems that involve repetitive or recursive structures. Recursive predicates can be easily defined in Prolog, allowing for elegant and concise solutions to problems that require repeated computations or traversals of data structures. For example, the following Prolog code defines a predicate to reverse a list using recursion:



This code defines a recursive predicate "reverse" that reverses a list by recursively splitting it into its head (X) and tail (Rest), reversing the tail, and appending the head to the reversed tail.

**1.4 Unification**

Unification is a central concept in Prolog, which involves matching and combining terms and variables to create a unified structure. Prolog uses unification to evaluate queries against predicates and to find solutions that satisfy the given constraints. Unification allows Prolog to perform powerful pattern matching and reasoning, making it a key feature that enables Prolog's expressive and flexible programming capabilities

**1.5 Built-in Parallelism.**

Prolog is inherently designed for parallelism, which makes it well-suited for concurrent and parallel processing. Prolog allows developers to define independent predicates and goals that can be evaluated concurrently, leading to efficient and scalable solutions for problems that can be naturally parallelized. Prolog also provides built-in predicates for synchronization and communication among parallel processes, making it a powerful language for developing concurrent and parallel applications. For example, the following Prolog code defines two independent predicates that can be evaluated concurrently:

Graphical user interface, text

Description automatically generated

These two predicates can be evaluated concurrently, leading to efficient and scalable solutions for problems that can be naturally parallelized.

**1.6 Built-in Inference Engine and Automated Backtracking.**

Prolog comes with a built-in inference engine, which is responsible for evaluating queries against the defined predicates and finding solutions that satisfy the given constraints. Prolog uses a process called "unification" to match and unify terms and variables in queries with predicates in the program. If a query cannot be fully satisfied, Prolog automatically backtracks and explores alternative paths to find other solutions. This automated backtracking capability makes Prolog highly efficient in searching through large solution spaces and finding all possible solutions to a problem.

Text

Description automatically generated

This code defines a rule "predecessor" that finds the predecessor relationship by recursively checking the "parent" relationship.

The above features have shown that Prolog is an interesting language, so how to install and use it?

**2. Implement Prolog language.**

**2.1 Choosing a Prolog Programming Environment:**

There are several Prolog programming environments available, such as SWI-Prolog, GNU Prolog, SICStus Prolog, and others. For this report, we will use SWI-Prolog, which is a widely used and open-source Prolog implementation available for various platforms including Windows, macOS, and Linux.

**2.2 Installing SWI-Prolog.**

To implement Prolog language in SWI-Prolog, we need to first install the SWI-Prolog system on our machine. The installation process may vary depending on the operating system, but generally involves downloading the SWI-Prolog distribution from the official website (<https://www.swi-prolog.org/Download.html>). In this report we will use Stable version.

Graphical user interface, website

Description automatically generated

Step 1: Choose version (we will use Stable release version in this report)

Graphical user interface, text

Description automatically generated

Step 2: Choose the version that is suitable for your computer

Graphical user interface, application, Word

Description automatically generated

*Step3: Run the exe file you downloaded and do personalization*

Text

Description automatically generated

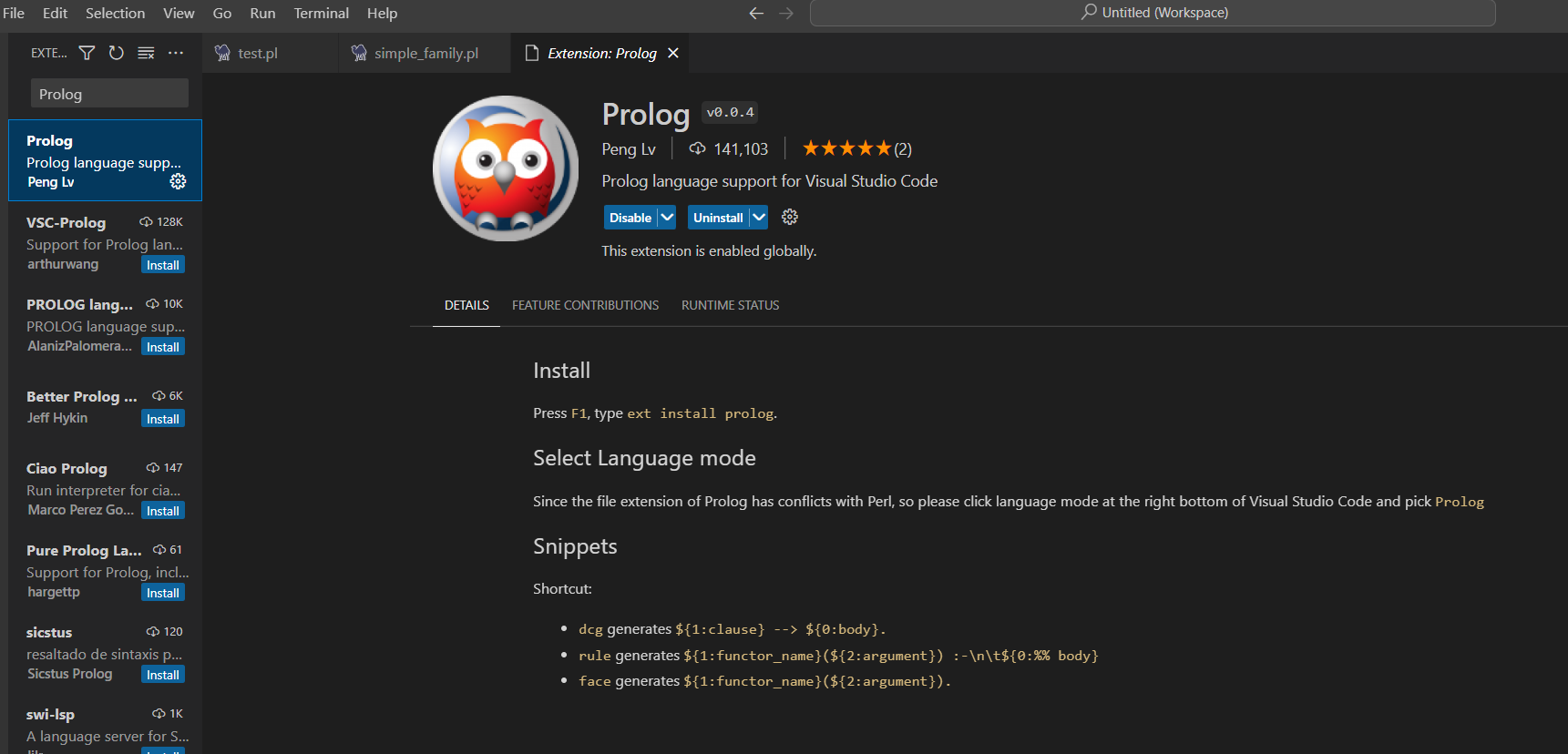
Step 4(Done): Run SWI-Prolog.

Text, letter

Description automatically generated

“Hello World” with Prolog

Implement on Visual Studio Code



Step 1: Install extension Prolog on Visual Studio Code



Step 2: Vào thư mục swipl/bin và lấy đường dẫn file swipl-win.exe

Graphical user interface, text, application, email

Description automatically generated

Step 3: Add path into Environment Variables

Text

Description automatically generated

Step 4 (Done): Open Terminal and type “swipl” command and and the same operation as on SWI-LOG.

Background pattern

Description automatically generated

Some of the most common symbols in Prolog.