# CS2910 Assessed Coursework 2

v2.0

This assignment **must be submitted** by 10am on the 8th of March 2024.

Feedback will be provided by 11th April 2024.

## 1 Learning outcomes assessed

This assignment assesses knowledge and understanding of finding paths using *uninformed search* implemented in Prolog. In particular, the outcomes assessed are:-

- knowledge and understanding of uninformed search strategies such as *depth-first* search;
- application of uninformed search strategies to search problems represented as trees and graphs;
- implementation of uninformed search strategies in Prolog;
- extensions of uninformed search strategies with a cost weighting.

## 2 Instructions

Submit this coursework using Moodle by clicking here. Your submission should consist of a `cs2910.zip` file, compressing a file named `cs2910CW2.pl`, containing all parts of the coursework.

It is your responsibility to check that the submitted file:

1. is named correctly;
2. has no syntax errors;
3. runs with SWI Prolog 7.6+;
4. is not empty; and
5. is not corrupted.

**Warning**: If any of the above conditions (1...5) is violated, your mark will be set to 0. You will be given an opportunity to resubmit for feedback purposes but the marks will **not** count for assessment. Submission after the deadline will be accepted but it will automatically be recorded as being late and is subject to College Regulations on late submissions.

**NOTE:**

All the work you submit should be solely your own work. Coursework submissions are routinely checked for this. Any assessment offence will be investigated subject to the College regulations.

# 3 Searching using Prolog

## 3.1 Finding a path

Consider the plan of a house as shown in Fig. 1. Write a Prolog description of this plan that allows a Prolog program search for paths between locations indicated on the plan. For example, if one is Outside and wishes to find a path to the WC, then a possible answer can be: Outside → Porch 1 → Kitchen → Leaving Room → Corridor → WC. Your program should

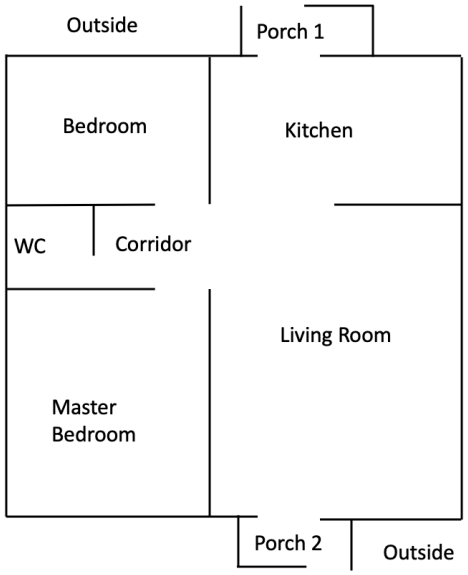


Figure 1: Top view of a two bedroom house showing how different locations in and out of the house are connected.

avoid loops and be as general as possible, in that, given any two locations representing an origin O and a destination D, it returns:

- a path P from O to D, if it exists;
- every possible path P from O to D, if there are more than one;
- meaningful error messages, if the wrong input is provided for O and D.

Comment the code submitted for this part to contain the name of the search algorithm that you selected to find the path. [**40%**]

## 3.2 Paths ending at a common destination

Define a Prolog program that searches bi-directionally from two origins O1 and O2 to meet up at a common destination D. It is up to you how you define that, perhaps by reusing parts of the program you developed for section 3.1, however this time you do not have to check that

the input parameters are correct, simply assume they are. Your program should combine a path P1 from O1 to D, with a path P2 from O2 to D, if they exist, to produce a combined path P, showing how you reach meeting point D from O1 and O2 respectively. You are expected to produce all possible combinations P1 and P2 to construct P non-deterministically, if we ask for more answers. Test your program using Fig. 1 by asking different queries and see whether your work produces the expected outcomes. Maximum marks will be obtained for also providing a program that returns only the shortest paths P1 and P2 combined to produce P. [**30%**]

## 3.3 Paths with Costs

Rewrite the problem description for the program you defined in section 3.1, to include the individual cost C (an integer describing the distance in meters) between locations of the house as shown in Fig. 2. Once you have a new representation that takes individual cost between locations into account, rewrite also you search program to calculate the total cost of a path, so that when you return a path P as a solution, you also return $C_{total}$. Your new program should also return every path, if there is more than one, returning the cheap ones first, and then the more expensive ones, in rank order. Then use your new program to define how to meet up from two origins O1 and O2 to a destination D (as in 3.2), only if the cost $C_{total}$ is the same from both O1 and O2 respectively.

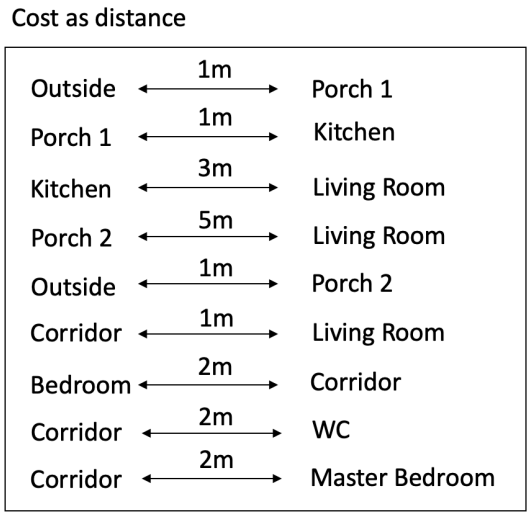


Figure 2: Cost between locations.

Test your program using Fig. 1 by asking different queries and see whether your work produces the expected outcomes. [**30%**]

# Marking criteria

- Full marks will be given for implementations that address the requirements of all the tasks and their sub-tasks as specified in this document.
- Marks will be allocated to the logic of the implementation strategies proposed as well as their implementation .
- Marks will also be allocated in solutions which show understanding of Prolog unification, especially on the use of lists, the use of existing primitives that manipulate and generate lists (like `append/3`, `findall/3` or `setof/3`), including the relevance of all these to the specific search requirements.
- Code quality: indentation, comments, variable naming, use of '_' variables, and appropriate use of Prolog control operators (e.g. the cut operator (!)).
- It is expected that **the files you will submit are created using Linux and not Windows**, as the marking will be done on a Linux machine. The code should run in `SWI Prolog` version installed on `linux.cim.rhul.ac.uk`. Implementations in any other Prolog or programming language will not be accepted.
- Your code should compile successfully for full marks. If part of your code does not compile, then wrap it in a comment of the form:

  ```
  /* Partial Code:
  ....
  End of Partial Code */
  ```

  and we will try to mark any logic that is relevant to the required task.