|  |  |
| --- | --- |
| [Staffordshire University Logo](http://www.staffs.ac.uk/) | Faculty of Computing, Engineering and Sciences |

# COSE60501 Advanced Programming Language Concepts

## Coursework 2015

This assignment seeks to address the learning outcomes 2 and 3 specified in the Module Specification.

These are as follows:

Outcome 2 : Express queries, facts and rules in a logic programming language.

Outcome 3 : Implement primitive recursion and highorder functions in a functional language.

**DEADLINE FOR SUBMISSION: 15:30 May 1st 2015**

**The Travelling Salesman Problem**

The travelling salesman problem is a classic combinatorial optimisation problem. The problem itself is straightforward, a travelling salesman has a set of locations to visit, which is the shortest route for him to take that visits each location and returns home?

LEEDS

LINCOLN

SHEFFIELD

LIVERPOOL

LEICESTER

LANCASTER

NOTTINGHAM

MANCHESTER

BIRMINGHAM

STOKE-ON-TRENT

Solving the problem is notoriously hard, as the number of possible routes and size of the problem increases greatly with additional of every location.

**Exhaustive/Brute force solution**

The simplest strategy for solving the problem is an exhaustive or ‘brute force’ search. This involves identifying and evaluating every possible route to find the one with the short distance. This solution guarantees that the shortest possible route will be found but at the cost of time and resources (memory & processing), especially as the size of the task increases exponentially with each additional location.

**Alternative methods**

It is obvious that many of routes evaluated by an exhaustive search will not be the optimum, e.g. the route in which the next location is always the furthest away. A search that uses some strategy or ‘rule of thumb’ to reduce the number of options to evaluate is called a heuristic search. These techniques usually involve some degree of compromise, trading the amount of time/processing for the accuracy/quality of the final solution.

**The Greedy algorithm**

The strategy for the ‘Greedy’ algorithm is to select the local/immediate optimum option at each decision point. For the travelling salesman this translates into always choosing the closest unvisited location for the next journey. This strategy means that the algorithm will typically complete in a fraction of the time required for the exhaustive search. However, by only considering local problems it is possible to embark on a path that is ultimately poor.

**The Problem Data**

Table 1 contains the data for the assignment, the distances in miles (by road) between ten of the largest cities in and around the midlands. Starting from ‘home’ in Stoke-on-Trent find the shortest route that visits each of the other cities once and returns back to home.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | BIRM | LANC | LEEDS | LEIC | LINC | LIVER | MANC | NOTT | SHEFF | **S-O-T** |
| BIRMINGHAM | 0 |  |  |  |  |  |  |  |  |  |
| LANCASTER | 129 | 0 |  |  |  |  |  |  |  |  |
| LEEDS | 119 | 88.3 | 0 |  |  |  |  |  |  |  |
| LEICESTER | 43.6 | 149 | 97.4 | 0 |  |  |  |  |  |  |
| LINCOLN | 98.6 | 152 | 71.6 | 54 | 0 |  |  |  |  |  |
| LIVERPOOL | 98.6 | 57.4 | 72.6 | 119 | 138 | 0 |  |  |  |  |
| MANCHESTER | 86.3 | 55.4 | 42.5 | 107 | 85.2 | 34.9 | 0 |  |  |  |
| NOTTINGHAM | 52.2 | 141 | 71.6 | 28 | 39.9 | 111 | 80.9 | 0 |  |  |
| SHEFFIELD | 85.3 | 93.3 | 35.5 | 64.2 | 48.6 | 77.1 | 37.9 | 38.8 | 0 |  |
| **STOKE-ON-TRENT** | 44.5 | 86.3 | 92.1 | 60.7 | 90.7 | 56.3 | 44.7 | 52.4 | 47.4 | 0 |

Table 1. Distances between the 10 cities.

**The Assignment Task**

The aim of the assignment is to test both programming skills and understanding of the languages and concepts. You are required to use two programming languages, one must be Haskell the other can be an imperative language of your choice (e.g. C, C++, Java, C#, Python).

**Programming tasks**

Basic - Implement an exhaustive/brute force algorithm to find the shortest route.

Intermediate – Implement the greedy algorithm.

Advanced – Research and implement and another heuristic search algorithm. This can be an improvement on the greedy algorithm (e.g. branch and bound) or a completely alternative approach.

**Evaluation and understanding tasks**

Basic – compare development and execution characteristics of the two implementations. For development, consider factors such as time, lines of code, languages features, etc. For execution consider execution speed, resource usage, etc.

Intermediate- Relate performance of each solution to the number and type of operations involved in finding the solution and the processes the language will undertake in completing these operations. For example, consider if the solution is compiled or interpreted and the impact of language features such a ‘lazy’ evaluation.

Advanced – Based on the analysis of results from different solutions (e.g. brute force, greedy, etc.) and larger problems (11 cities, 12 cities or more) consider which language might provide the better overall solution to the travelling salesman problem (consider development time, resources requirements, speed, scalability, etc.).

**The Artefacts**

To achieve a pass you are required to produce two **working** artefacts (one in Haskell the other in your selected language). The submitted code should be **fully commented** and submitted with the assignment submission. In your comments state any assumptions made about the problem and your solution.

All language features discussed in the classes should be used where appropriate, and credit will be given for use of concepts which go beyond the techniques discussed in class if applied correctly and fully justified.

**The Report**

The report should be no more than 2000 word, and represent a comparison of the experience of using the two selected languages to solve the problem. The comparison should be based on the following headings.

* Brief discussion of the problem and the solution(s) implemented.
* Methods of testing, results and discussion of the comparison.
* Conclusion about the languages comparison

A conclusion should discuss the relative advantages and disadvantages of the programming paradigms discussed and a reflection on the experience.

**Marks and marking Scheme**

The overall marking scheme is given in Table 2 and the requirement for each of the grades is given in Table 3.

With respect to the code, the artefacts presented should work (at least to some degree) and represent a solution to the problem.

Two solutions should be presented which at the very least exploit where appropriate language features discussed during the lectures.

Additional credit will be given for evidence of the use of features not covered during the formal teaching, although this is not a requirement. The code should be intelligible enough to someone with a basic understanding of the selected language such that they can follow what has been attempted/achieved and comments should be used extensively to highlight any assumptions, or to clarify the function/purpose of the code. What cannot be understood, will not be marked!

With respect to the report, there should be an introduction in which the context for the work is set with reference to textbooks on the languages selected and where appropriate journal articles.

The work is required to be original in the sense that it is the student’s own work and also there should be evidence of reading around the topic given incorporating material not necessarily included in the lecture notes. Evidence of this reading and its understanding will form part of the marking criteria.

The report should contain a conclusion in which reflection is undertaken both on the student’s fulfilment of the assignment task and consideration of whether the languages selected were appropriate for the problem. This should involve careful thought, and again evince some evidence of reading.

The report should use Harvard Referencing, be appropriately structured and presented to a standard commensurate with your level of study.

The marking for each component is as follows:

|  |  |
| --- | --- |
| Artefact A | |
| Overall Functionality | 5 |
| Use of language features | 10 |
| Program Design | 5 |
| Code Readability | 5 |

|  |  |
| --- | --- |
| Artefact B | |
| Overall Functionality | 5 |
| Use of language features | 10 |
| Program Design | 5 |
| Code Readability | 5 |

|  |  |
| --- | --- |
| Report | |
| Context Setting | 10 |
| Originality of approach | 10 |
| Evidence of reading, understanding and applying | 15 |
| Critical Evaluation | 15 |

Table 2: Marks Breakdown

|  |  |
| --- | --- |
| First 70%+ (A) | Well-constructed implementations of the  problem, accompanied by a fully-referenced report which shows evidence of a deep engagement with the relevant literature and the theoretical issues. |
| Upper Second 60 - 69% (B) | An assignment which, in its totality (artefacts and report), falls short of those criteria required for a First. For example slight deficiencies in the implementation, or the report. |
| Lower Second 50 – 59% (C) | An assignment which, in its totality (artefacts and report) falls short of the requirements for an Upper Second. For example significant deficiencies in the implementation or the report, although the approach is basically sound. |
| Third 40 – 49% (D) | An assignment which, in its totality (demonstration and report) falls short of the requirements for a Lower Second. For example significant deficiencies in the implementation and the report, but where there is evidence of merit in the approach taken. |

Table 3: Guide to standard expected by grade

**Submission instructions will be provided on Blackboard**.