

Faculty of Applied Sciences  
Bachelor of Science in Computing

**COMP490 Final Year Project  
Progress Report**Academic Year 2022/23

|  |  |
| --- | --- |
| Parsons problem generator and solver | |
|  |  |
| Project number: | 19 |
| Student ID: | P-19-0834-5 |
| Student Name: | Jane Liu |
|  |  |
| Supervisor: | Philip Lei |
| Assessor: | Charles Lam |
|  |  |
| Submission Date: | 24/11/2022 |

Declaration of Originality

I, Jane Liu, declare that this report and the work reported herein was composed by and originated entirely from me. This report has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

Text

Description automatically generated with medium confidence

Jane Liu

02/11/2022

Abstract

This template file provides the Word styles for writing the Final Year Project.

Text highlight in green are instruction or hints. Text highlight in gray are sample text to demonstrate formatting. The following paragraph is an example.

Sample text sample text Sample text sample text Sample text sample text Sample text sample text. Sample text sample text Sample text sample text, Sample text sample text Sample text sample text.

There are also some placeholder highlight in red. Change them to your own information, e.g. your name and project title.

In any submitted report, you must delete or replace all the colour text.

Table of Contents

[1 Introduction 6](#_Toc118402819)

[1.1 Objectives 6](#_Toc118402820)

[1.2 Risk Assessment 6](#_Toc118402821)

[1.3 Summary 7](#_Toc118402822)

[2 Background and Related Work 8](#_Toc118402823)

[2.1 Domain 1 8](#_Toc118402824)

[2.2 Domain 2 8](#_Toc118402825)

[2.2.1 A Subtopic in Domain 2 8](#_Toc118402826)

[2.2.2 Another Subtopic in Domain 2 8](#_Toc118402827)

[2.3 Related Work 9](#_Toc118402828)

[3 Completed Work 10](#_Toc118402829)

[3.1 Parsons Problem Analysis 10](#_Toc118402830)

[3.2 Second Topic 11](#_Toc118402831)

[4 On-going and Future Work 12](#_Toc118402832)

[4.1 First Topic 12](#_Toc118402833)

[4.2 Second Topic 12](#_Toc118402834)

[5 Conclusion 13](#_Toc118402835)

[References 14](#_Toc118402836)

Table of Figures

[Figure 1: Probability impact matrix before proposed solution 1](#_Toc454957831)

List of Tables

[Table 1: Table of prioritized risk 1](#_Toc449539554)

# Introduction

Introduction must include the following:

* Background and motivation. Problem. Related works
* Project description
* Objectives
* Main tasks

Refer to the Writing Guide and the Writing Workshop for more detail on the content requirement for each chapter.

## Objectives

Write one paragraph to state the aim or goal of the whole project. Then break down the goal into 4-6 SMART objectives.

The objectives of this project are:

* Sample objective 1
* Sample objective 2

## Risk Assessment

Introduce the main risks of your project in this intro paragraph

Table 1: Table of prioritized risk

|  |  |
| --- | --- |
| Priority | Risk Identifier and Description |
| 1 | Risk 1: short description |
| 2 | Risk 2: short description |
| 3 | Risk 3: … |
| 4 | Risk 4: .. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Probability** | High |  |  | Risk 1 |
| Medium |  | Risk 3 | Risk 2 |
| Low |  |  |  |
|  |  | Low | Medium | High |
|  |  | **Impact** | | |

Figure 1: Probability impact matrix before proposed solution

## Summary

The summary should be finished like this: This report is organized as follows: Chapter 2 introduces the background of our work. Chapter 3 presents our design approach. Chapter 4 shows the implementation details….

# Background and Related Work

In this chapter, you provide background information for readers to help them understand your project. There may be more than one sections on background domain.

This chapter also provides detail about related work. In chapter 1, you should have mentioned some related works and explain how your project is related to them when you discuss relevancy. If you want to provide detail of related works, include them in this chapter.

Call this chapter “Background” if all related works have been described in Chapter 1.

## Domain 1

Sample text sample text Sample text sample text Sample text sample text Sample text sample text. Sample text sample text Sample text sample text, Sample text sample text Sample text sample text.

## Domain 2

Sample text sample text Sample text sample text Sample text sample text Sample text sample text. Sample text sample text Sample text sample text, Sample text sample text Sample text sample text.

### A Subtopic in Domain 2

Sample text sample text Sample text sample text Sample text sample text Sample text sample text. Sample text. A sample list below:

* Sample list item
* Sample list item

### Another Subtopic in Domain 2

Sample text sample text Sample text sample text Sample text sample text Sample text sample text. Sample text.

## Related Work

Give a brief description of related works. May be omitted if enough detail of related works are already covered in Chapter 1.

Sample text sample text Sample text sample text Sample text sample text Sample text sample text. Sample text sample text Sample text sample text, Sample text sample text Sample text sample text.

# Completed Work

This chapter explains the completed work up to now including three different parts. Specifically, the first part illustrates the new ways of using Parsons problem to fit the exercises in CS2. The next part shows the corresponding data modelling to support these new ways. Finally, the implementation details of a prototype of Parsons problem are covered.

## Parsons Problem Analysis

Previous type of Parsons problem has shown its own advantages in exercises of CS1. However, since there are plenty of differences between the exercises in CS1 and CS2, it is not so proper to apply Parsons problem directly in CS2 without tailored improvement. Thus, some new ideas should be introduced to Parsons problem. The limitation of the previous Parsons problem and new ideas will be discussed in the following paragraphs by comparing the difference between CS1 and CS2.

Unlike CS1, the codes in CS2 are much more complex and abstract in the following three general aspects. To be more specifically, students are supposed to not only understand the meaning of every lines but also be able to organize these single lines into larger blocks like methods, classes, even algorithms in order to have a deeper understanding for the features of data structures, the recursive solving method and the algorithms analysis. In addition, codes are no longer the only points covered in Parsons problem. Some new abstract elements like pseudocode and algorithms analyze should also be involved in exercise of CS2. Finally, high comparing ability is required in CS2 since there are plenty of similar concepts like stacks and queues, different types of trees, and different algorithms to solve the problems.

The difference between CS1 and CS2 courses mentioned above leads to the deficiencies in applying the previous type of Parsons problem. To show these limitations in detail and introduce the improvement aiming at these limitations, the exercises in the popular textbook *Data Structures and Algorithms in Python* [1]were analyzed and five types of common questions were identified.

### Object Oriented Programming

Object-oriented programming is a programming model based on objects, which includes attributes and methods [2]. In CS1, since all the programming parts are Structured Programming, the object-oriented programming concept has never been involved. However, in CS2, this concept becomes one of the focal points since it is the foundation to build different data structures. But after introducing this concept, Parsons problem for data structure questions cannot have unique answers since the methods in the classes can change their positions without effecting on the correctness of the results (for example, Figure (Changing the methods positions does not affect the correctness of the results) ).

To have a unique answer, some paragraphs describing the functions of the methods should be listed in a specific order in the question description, and students are asked to build methods one by one in a matching order. For example, in Figure 2, the left part shows the description of methods of an abstract data type stack in a specific order, and the corresponding methods are implemented in a matching order in the right part. In this way, a unique order of answers is defined by design. Thus, Parsons problem can not only fit the object-oriented programming type of questions but also keep some of its reordering features at the same time. More importantly, this new design idea enhances students’ understanding of the whole method instead of just every individual line. Further, by this design, students also can have a general idea about what features of a specific abstract data type (ADT) should have [1].

Graphical user interface

Description automatically generated with low confidence

Figure 2 The description of methods in a stack (left) and the corresponding codes (right)

Another way to handle this problem is to only check the availability of methods and the contents of methods but ignoring the positions of the methods. This new design is significantly different from the previous type of Parsons problem. Specifically, in the previous type of Parsons problem, the interchangeable lines are combined to a larger block to have a unique answer [3]. In this way, it ignores the smaller parts (lines), and focuses on the bigger one (blocks). By contrast, the new design looks like an opposite way comparing with the previous design. It ignores the bigger parts (order between different methods) and focus on the smaller parts (lines inside the methods).

(Example: two different work procedure)

### Algorithm Analysis

Algorithm analysis, which is a core part of CS2, is to use running time to evaluate whether a data structure or an algorithm is efficient or not [1]. Since the growth rate of running time as a function of the input size n is vital for algorithm analysis, big O, which shows the major parts affecting the growth, is introduced in algorithm analysis [1]. There are seven common big O classes in CS2 (the constant function, the logarithm function, the linear function, the N-log-N function, the quadratic function, the cubic function and other polynomials, and the exponential function) [1] so it is strenuous to ascertain which functions are the correct functions to describe the efficiency of the specific data structures and algorithms. By contrast, the algorithm analysis in CS1 is much more straightforward since only four functions (the constant function, the linear function, the quadratic function, and the cubic function and other polynomials) are used and the functions used are only lying on whether there are loop structures or nested loop structures. For this reason, the algorithm analysis parts are always ignored in CS1 so the Parsons problem for CS1 certainty cannot handle this algorithm analysis questions, which is another drawback for using previous types of Parsons problem directly in CS2.

To involve the algorithm analysis, only operating the code cannot satisfy the situation anymore. Thus, a new component – comment – is introduced in Parsons problem. To be more specific, the comments are used to provide different big O classes for choosing. Students need to select the correct corresponding big O class comments and insert them into codes to do algorithm analysis of the whole code. Unlike the previous reordering questions, this question is more like a multiple-choice question since students can put the comment in wherever they like. Furthermore, this new idea also can be expanded in two ways. One way is to provide not only big O but also the described reasons to get big O. In this way, students can have a deeper understanding of algorithm analysis instead of just guessing big O classes according to their feelings. Another way is to let students match different big O classes with different parts in codes. This way can be used to track the change points of big O classes (like outer for loop and inner for loop) or it can be used to compare different methods in a data structure and let students know the characteristics of a data structure (For example, a heap is quick to get the min value but it is slow to add nodes, shown in Figure 3).

Table

Description automatically generated

Figure 3 Performance of a heap-based priority queue

### Recursion

Recursion is to solve a problem by solving a subproblem that has the same structures as the original problem. It is achieved by calling the function itself, passing smaller argument value to build subproblems, finding the answers of subproblems and using these answers to solve the original problem. Unlike in CS1, the recursion method is called inside instead of outside of the method body, which requires students with deeper understanding in the whole picture of entire methods besides concrete lines in the methods. For this reason, it leads to a dramatic increase in the difficulty of problems and the previous type of Parsons problem cannot have a significant effect on reducing complexity. Concretely, the previous type of Parsons problem reduces the level of difficulty by providing some code fragments for students to read. However, in the recursion type of question, it is arduous to understand the meaning of every line of code even if the whole code is given in order (like the recursion example code of the Tower of Hanoi in Figure 4 [4]). Thus, the assistance coming from code reading of Parsons problem is notably shrunk, and it is almost like letting students write code directly. That is why the previous type of Parsons problem needs to be improved to give students more hints in solving recursion questions.

Text

Description automatically generated

Figure 4 The recursion example code of the Tower of Hanoi

According to the suggestions for designing recursive algorithms provided by the textbook, it is beneficial to find a few specific subproblems which have the same structures as the original problems [1]. To achieve the suggested way, a new pre-scaffold way – organizing in several steps – is introduced into Parsons problem. To be more specific, one recursion question is divided into some steps. The first step is to let students reorder the code of base cases (the end point of every recursion chain [1]). Subsequently, change the values of parameters used in this recursion algorithm to have other subproblems. And in this step, the previous base cases should also be involved. In other words, students need to build these subproblems by calling the method in the base cases. In the following step, students should choose the correct method headers with proper signatures. In the final step, students are required to reorder the real recursion codes with all the previous codes as references. This new pre-scaffold way can be partially helpful for students to solve the recursion problems. In this new way, students are provided with more hints (like subproblems) than directly solving recursion questions. Thus, they are less like to be all at sea.

An example

### Data Structures and Algorithms

One of the characteristics of code in CS2 is similar to each other. For example, the codes can have the same data structures but are implemented in different ways (like the array-based stack and linked-list-based stack in Figure 5) or different solving algorithms for the same problems (like bubble sort and selection sort for sorting numbers problem). Although it is ok to use the previous types of Parsons problem individually for each code, it would be more worthwhile to help students to compare these similar codes and consolidate the difference and similarity between these codes. Therefore, the students can distinguish them and have a better understanding.

Table

Description automatically generated

Figure 5 The array-based stack and the linked-list-based stack

To compare the same data structures implemented in different ways, the transforming learning method is used. In other words, two different codes with some common parts are provided, and students need to choose the common parts from one of the codes using these common parts to build another one. In this way, students can understand new knowledge by transforming what they have learned before. Taking the two implementation ways of a stack as an example, students can drag and drop the method headers in an array-based stack as some components to build a linked-list-based stack. In such a manner, students can have a general idea about what the common parts are (the methods supported in a stack abstract data type) and what differences are (implementation details caused by array structure or linked list structure). Besides, the completed ordered array-based stack codes also can be hints to support students to build the linked-list-based stack.

An example should be added here

To compare different algorithms of the same question, two algorithms are mixed by using distractors, and students are supposed to distinguish and order each of them from the mixing code pool. To be more specific, unlike previous distractors, which are all used to show some incorrect or improper code [5], a group of correct complete codes of an algorithm is introduced as distractors in order to mix with another algorithm. In other words, the jumbled code fragment pool includes two different correct algorithms, and students should pick up, rearrange, and submit the code fragments in these two algorithms separately. This setting method is significantly applicable in the CS2 since there are a lot of questions having this characteristic – one question with several solving algorithms, for example, bubble sort, selection sort, insertion sort for sorting numbers, and breadth-first search, depth-first search for searching. In this way, it can help students to distinguish similar algorithms in the same categories mentioned before, which is worthwhile when students have learned more than one algorithm and begin to use them motley because of blurry memory.

An example should be added here

## Highlight of Analysis

### Balance the degree of difficulty

The key function of Parsons problem is to add a preparation stage before students actually begin to write codes. This preparation stage is achieved mainly by reducing the degree of difficulty caused by directly writing codes. In other words, one of the key issues of Parsons problem is to find a balance point in the degree of difficulty, which is easier than writing codes but still valuable to practice. To fulfill this demand, three different methods (selecting difficulty level, pre-scaffold, and content) are introduced in the following paragraph.

In the previous Parsons problem, no matter whether it is pre-scaffold or student-scaffold, the problem always is predefined with a fixed difficulty level, which means that students cannot ask for further help with additional hints if they meet problems in solving this Parsons problem. Since the codes in CS1 are quite simple, this problem may not happen frequently, and it may not affect students solving problems. However, due to the improvement in the complexity of the codes in CS2, this problem begins to become a huge issue. To handle this issue, multiple difficulty levels are set, and students are allowed to switch to an easier version of this question halfway. For example, in Figure [], three different difficulty levels are set. Specifically, Level 0 is to just cut codes in lines and allow students to reorder them. Subsequently, in Level 1, the headers of methods and predefined description comments are placed in the correct positions for hints. In Level 2, the codes are grouped by methods so that students can only reorder smaller parts of codes for every method instead of selecting from the whole mixing codes pool. In Level 3, some codes in methods are fixed as context and only some core parts need to be reordered. If students find the present version of Parsons problem too difficult, they can change to an easier version.

[An example needs to be added]

Pre-scaffolding is to give students some structures for hints. This kind of method to handle the difficulty has already been used in the existing Parsons problem, like providing loop structures. This method is also used in CS2 courses but with some extension. To be more specific, pre-scaffolding can be achieved by separating codes by different methods like the example shown before. In addition, pre-scaffolding also can be achieved by separating codes into different steps like the example shown in recursion questions. Finally, this method also can be achieved by separating codes according to different subgoals helping students to divide a huge, complex problem into smaller simple questions.

[An example needs to be added]

Context is used to provide some ordered codes as information and let students only reorder some smaller parts in the middle of these ordered codes. Context can be done a step further, which means that students only need to insert some key code fragments into the right location of the whole code file. And to keep the question valuable, some wrong codes are also provided to confuse students. This kind of question is extra useful in iterator and recursion questions.

[An example should be added]

### Game-like Experience

To make the procedure of solving questions more interesting, some game design ideas are added to this project. Not only game-design elements like colorful icons and rewards are included, but also some design model liking freezing time before asking another feedback are used. Like the following example…

[An example should be added]

# On-going and Future Work

Write an introduction paragraph to delineate the content and logic flow of this chapter.

* Describe partially done works
* Include a Gantt chart as evidence of effective project planning for the 2nd semester
* Show Clear idea of what to do to complete the project

## First Topic

Sample text sample text Sample text sample text Sample text sample text Sample text sample text. Sample text sample text Sample text sample text, Sample text sample text Sample text sample text.

## Second Topic

Sample text sample text Sample text sample text Sample text sample text Sample text sample text. Sample text sample text Sample text sample text, Sample text sample text Sample text sample text.

# Conclusion

Reflect on the progress of the project. Can use first person pronoun to write.

Content may be moved to the Reflection appendix in the Final Report.

Sample text sample text Sample text sample text Sample text sample text Sample text sample text. Sample text sample text Sample text sample text, Sample text sample text Sample text sample text.

References

[1] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. Structure and Interpretation of Computer Programs. MIT Press, Cambridge, Massachusetts, 1985.

[2] Robert Baumgartner, Georg Gottlob, and Sergio Flesca. Visual information extraction with Lixto. In Proceedings of the 27th International Conference on Very Large Databases, pages 119–128, Rome, Italy, September 2001.Morgan Kaufmann.

[3] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. Cognitive Science, 9(2):171–216, April–June 1985.

[4] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. Journal of Computer and System Sciences, 64(3):579–627,May 2002.

[5] Georg Gottlob. Complexity results for nonmonotonic logics. Journal of Logic and Computation, 2(3):397–425, June 1992.

[6] Hector J. Levesque. Foundations of a functional approach to knowledge representation. Artificial Intelligence, 23(2):155–212, July 1984.

[7] Hector J. Levesque. A logic of implicit and explicit belief. In Proceedings of the Fourth National Conference on Artificial Intelligence, pages 198–202, Austin, Texas, August 1984. American Association for Artificial Intelligence.

[8] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. Journal of Artificial Intelligence Research, 12:271–315, 2000.

[9] Ivan Marsic. A short guide for writing a thesis. http://www.ece.rutgers.edu/~marsic/thesis-guide.html, 2004 [Mar. 6, 2014].

[10] Matlab documentation. http://www.mathworks.com/help/?s\_tid=hp\_ff\_s\_doc [Mar. 6, 2014].

[11] George Sparling. Spacetime is spinorial; new dimensions are timelike. arXiv:gr-qc/0610068v1, 2006.

[12] Ryan Rifkin. Everything old is new again: a fresh look at historical approaches in machine learning. Ph.d thesis, MIT, 2002.