

Edge Hill University

The Department of Computer Science

**CIS2147**

**Programming Languages: Theory to Practice**

Level 5

Coursework 2

2021/2022

### Module Leader: Dr Ardhendu Behera

🕿 01695 65 7270

**Email:** [**beheraa@edgehill.ac.uk**](mailto:beheraa@edgehill.ac.uk)

## *Administrators:*

### 🕿 01695 65 7603

|  |
| --- |
| Coursework 2 |

Weighting: **40%**

Hand in date: **7th January 2022 (by 12:00 noon)**

Feedback date: **Within 4 weeks after submission date**

Learning Outcomes Assessed:

LO1 Compare and contrast the broader issues surrounding programming paradigms and styles in the context of application development.

LO2 Systematically evaluate approaches to implementing solutions to software development problems through the analysis of the software requirements and the paradigms available to them.

LO3 Apply appropriate programming paradigms in the design and implementation of a software solution.

LO4 Define the role of automated test tools in the testing of software and apply appropriate techniques in order to utilise such tools.

|  |
| --- |
| Introduction |

This assignment is compulsory as it is a major part of the formal assessment. You will be working to complete the design and implementation of a software system, so it is important that you should develop effective strategies for managing your time. You should refer to the guidance in the module handbook regarding the documentation of your coursework. Read this document carefully and make sure that you are clear about what you have to do, and what you have to hand in, before you attempt the assignment.

The aim of this assessment is to give you the opportunity to experiment further to develop your programming skills, alongside developing your understanding of programming paradigms. You will undertake background research and then implement different solutions to a problem to be able to evaluate the use of different programming paradigms and styles.

You could submit drafts and/or demonstrate your code to the module tutor during practical sessions for formative feedback prior to the submission date.

|  |
| --- |
| Background |

For this task, you will be required to complete the required implementation the provided Binary Tree (BT) program code (MyBinaryTree.cs) to build the below tree. The complexity of the Binary Tree can be increased by adding more data points (nodes). You need to complete the methods (highlighted in the provided skeleton code) by adding your codes. Look for the comments “Add your code here”. The task also requires the implementation of the below functionalities:

Binary tree example.

1. **In-order**, **Pre-order** and **Post-order** traversals of the above given Binary Tree.
2. In the above tree, search the item 20 using *depth-first-search* (DFS) algorithm (refer to the lecture slides for the pseudocode) using a **Stack**. You could implement any of the DFS algorithm (i.e. either In-order or Pre-order or Post-order). In this exercise, you are allowed to use the existing in-built stack library in C# (i.e. use “Stack stack = new Stack();” to create an empty stack). While searching the element print the item in the visited node. If the element is found print the path from that element to the root node.
3. Amend the above program to search the item 30 using *breadth-first-search* (BFS) algorithm (refer to the lecture slides for the pseudocode) using a **Queue**. In this exercise, you are again allowed to use the existing in-built queue library in C# (i.e. use “Queue q = new Queue();” to create an empty queue). While searching the element print the item in the visited node. If the element is found print the path from that element to the root node.
4. This exercise was to use a depth-first search (DFS), using a stack to find the node in the binary tree containing item 20 and printing each item within each visited node and once the element was identified then print the path back to the root node. The task was broken down in steps on paper as follows:



1. Use recursion instead of stack or queue to search both items 20 & 30.

Make sure your program runs correctly. You should give attention to the “NULL” assignments to the nodes that can make your algorithm in giving segmentation fault.

|  |
| --- |
| Tutor Details |

| **Module Leader**: Ardhendu Behera, BEng (Hons), MEng, PhD, FHEA | |
| --- | --- |
| **Location:** Technology Hub **Office:** THF12  **Direct line:** 01695 65 7270 | **e-mail:** [Beheraa@edgehill.ac.uk](mailto:Beheraa@edgehill.ac.uk) |
| Initial Planning | | |

### Contents of the Project Report

This section briefly describes the different elements that make up a typical Project Report. The general requirement is that you write up the project as a scientific experiment; after all, you are experimenting with different ideas of coding. You should be able to use the document you produce as a reference guide in future modules/work.

The main purpose of the coursework 2 is to produce software, building upon a range of appropriate programming skills.

The **Report** should briefly list the features in the work produced:

* **Description:** Which features are implemented, attempted, or considered? Be precise: if only part of a feature is implemented, say which parts are implemented and which are missing. How was the feature designed and implemented? Are there alternatives and what are the considerations? Provide background information based on academic sources.
* **Screenshots**: Provide as many screenshots of the output as necessary to demonstrate all aspects of the implemented or attempted features.
* **Implementation:** Code and comments.
* **Evaluation**: Report how you evaluated whether the task was completed to specification: tests conducted, problems encountered, solutions, remaining problems.

**References**: Academic sources include research articles and books written by and for academics. Use a referencing tool to ensure that you use correct British Harvard style.

<https://www.edgehill.ac.uk/ls/uni-skills/?tab=referencing>

### Academic Writing

The report should be a description of what you have achieved, not a story. The report should be written in the third person, so ‘I’ should not be used. Use a spellchecker to ensure that the spelling and grammar are correct.

<https://www.edgehill.ac.uk/ls/uni-skills/?tab=academic-writing>

|  |
| --- |
| What you should submit |

You should submit in *electronic format* through the university VLE, Blackboard:

* Your documentation (the Project Report)

The report should be fully referenced with background research and show that a proper process has been followed for the development of the binary tree: problem analysis, requirements gathering, program design, search algorithms, implementation, testing, and documentation completion. It should also include a critical review of the approach taken and a justification for the decisions made during the design process.

* The complete source code.

The developed source code should be well commented to explain the purpose of each part, how it is achieved, what its input and output is, and where the key decisions are made. To facilitate marking, you can add a *readme* text file, detailing how to run your code.

A drop box will be set up near to the deadline to allow you to submit them both. Remember that you should leave enough time to upload your work.

Through Blackboard, it is suggested you also keep a **week by week diary** (blog) detailing what you have done, what worked, what didn’t and how you are going to approach the work the following week. Discussion boards will also be made available on Blackboard and credit will be assigned for active participation in discussions and offering solutions to problems posted.

The report will be marked as an individual piece of work.

If you have any further questions/queries, please contact the Module Tutor during the class.

|  |
| --- |
| What will be assessed |

1. Report submission
   * Correct BCS structure
   * Introduction - Outlining problem including the potential outcome is expected.
   * Background research - Information surrounding the areas of the task (extra reading expected)
   * Mini project - Source code of the project including edited screenshots, videos, images and commentary
   * Comments in the code which clearly define the role of the code.
   * A bibliography of resources used.
   * Fully documented testing strategy and a discussion of the outcomes obtained.
2. Individual critique/reflection discussing how the task progressed, what were the issues that arose, how you overcame any issues and what would you do differently.
   * Participation in the discussion groups on Blackboard

| Grade (Mark Range) | Build (60%) | | | | Report (40%) | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Features (40%) | Software Design and Quality (20%) | Source Code Organisation and Quality (20%) | Source Code Documentation (20%) | Structure and Content (50%) | Test Documentation (30%) | Bibliography (20%) |
| Fail (0-29) | Nothing submitted or rudimentary work addressing none of the features. Unsuccessful attempt at implementing most of the functionality. | Design and build follows none of the concepts from lectures and portfolio. A small number of program components may be represented by designs, but designs contain serious errors. | Major technical flaws make it impossible to understand the code.  No systematic approach to build. | The code has not been commented in any way. No supporting commentary submitted. | Nothing submitted or substantially plagiarised. No evidence of reflective comments. | No evidence of testing. | No bibliography. |
| Narrow Fail (30-39) | Structure for at least one of the features exists, but no functionality has been implemented. A limited attempt has been made to explore alternative approaches to the program build. | Design and build follows some concepts from lectures and portfolio but contains significant issues. A small number of program components may be represented by designs, but designs contain a number of errors. | Code is readable, but without any apparent structure. No systematic approach to build. | There are occasional comments in some files. Supporting commentary identifies a small number of components. | Report contains a basic structure but no or very limited amounts of content have been added. No evidence of reflective comments. | An attempt is made to testing, but no details are provided. The procedure does not clearly relate to the problem being considered or the goals of the application. | Very limited number of sources in the bibliography. |
| Pass (40-49) | One of the required features implemented. An attempt has been made to explore alternative approaches to the program build, but may contain major errors. | Design and build generally follows key concepts from lectures and portfolio but may contain major issues. A number of program components may be represented by designs, but designs contain a number of errors. | Code is indented correctly, with easily identifiable nesting. There is some evidence that the approach to the build is systematic | The code is very loosely commented. Supporting commentary identifies a number of components, but contains major issues in identifying the role/purpose of those components in the program. | Report is descriptive. An attempt at documenting some of the required processes has been made, but contains flaws. Limited evidence of reflective comments. Little evidence of additional reading. | Report contains a brief description of manual testing. The presented test plan is related to the software. | Bibliography is in Harvard style. |
| Good (50-59) | One of the required features implemented. An attempt has been made to explore alternative approaches to the program build, but may contain minor errors. | Design and build mostly follows key concepts from lectures and portfolio but may contain minor issues. Designs are provided for most of the program components. | Variables, Functions, and Classes are appropriately named. There is evidence that the approach to the build is systematic | Most of the code has been commented. Supporting commentary identifies most of the components, but contains major issues in identifying the role/purpose of those components in the program. | Report is an overview discussion. An attempt at documenting some of the required process has been made and is accurate. Some evidence of reflective comments. Some evidence of additional reading. | Evidence of a partial test-plan being created and utilised. Analysis demonstrate limited depth of understanding. | The report is weakly referenced. |
| Very Good (60-69) | Two of the required features implemented. An attempt has been made to explore alternative approaches to the program build, but may contain minor errors. | Design and build follows key concepts from lectures and portfolio. Designs are provided for most of the program components. | A consistent coding style is used throughout. There is evidence that the code is organised and the build is systematic. | The code is fully documented. Supporting commentary identifies all of the components, but contains minor issues in identifying the role/purpose of those components in the program. | Report covers all required sections. All required process have been documented. Evidence of reflective comments and additional reading. | Evidence of a test-plan in line with the relevant testing strategies/tools but contains a small number of minor issues. Analysis demonstrates some depth of understanding. | Most arguments are supported by the relevant references. |
| Excellent (70-84) | Three of the required features implemented. An attempt has been made to explore alternative approaches to the program have been successfully implemented. | Design and build closely follows key concepts from lectures and portfolio. Designs are provided for all of the program components. | Where possible code is re-used, minimising duplicate code. The code is organised and the build is flawless. | As Good and the comments are succinct and relevant. Supporting commentary identifies all of the components, but contains a small number of minor issues in identifying the role/purpose of those components in the program. | As Very Good and all sections and process documentation is very detailed. Very good evidence of reflective comments and additional reading. | Evidence that the software has been tested as a complete system in line with the test-plan consisting relevant testing strategies/tools. Analysis demonstrates depth understanding of issues and critical evaluation of key points. | All arguments are supported by a relevant reference. |
| Outstanding (85-100) | All four features implemented. An attempt has been made to explore alternative approaches to the program have been successfully implemented. | Design and build represent unique solution to problem. Designs are provided for all of the program components with evidence of supplementary reading. | Code complexity has been minimized. The code is well-organised and the build is perfect. | As Excellent and higher-level documentation blocks added. Supporting commentary identifies all of the components. | As Excellent and the report shows extensive reflection on the work conducted with substantial evidence of additional reading. | Automated testing techniques have been used in line with the complete test-plan. Analysis develops a unique insight or offers original interpretations of findings. All findings are supported by relevant theory. | All arguments are supported by more than one relevant reference. |