**ASSIGNMENT BRIEF**

Programme: Games Development

Module code/title: 2CB101

Assessment mode: Portfolio

Assessor(s): Andrew Guest

|  |  |
| --- | --- |
| **DEADLINE** | Noon Mon 13th Jan 2020 |
| **HOW SUBMITTED** | Electronic through gitlab |

|  |  |
| --- | --- |
| **Overview**  *Please ensure you read all of this document. Page 3 gives a breakdown of the marking for this module. Page 4 gives a detailed breakdown on how the university grades different concepts.*  The assessment for this module consists of a C programming assessment and an Assembly language assessment. For each of these assessments you have to implement a code solution to a problem and write a document describing how the solution works. | |
| **1 – C – Storing Data In A Binary Tree**  Write a c program that will load the data from a data file (provided on Moodle). Your code should load the data in to a binary tree, inserting the data in the correct places within the tree. It should output a display showing the nodes as described below. It should store the data in a new file in sorted order.  Node Diagram shows each node in the following format  Key (Left:Key)(Right:Key)  Displaying each node in traversal order.  Write a document describing how your code works. You should pay special attention to what is stored where in memory and when. The lectures give examples of how this can be described. (1000 words maximum) | |
| **2 – Assembly Language vc C**  For this assessment you will have to create a C program and an Assembly Language program that do the same thing. You will use the GNU debugger to analyse both programs in action and compare them, paying careful attention to how the registers are used. C & Assembly Programs Both programs will have the same functionality. The will begin by outputting “FizzBang Demo” as a line of text. The program will then loop from 1 to 20, and for each number output the “Fizz” if the number is divisible by three, “Bang” if it is divisible by five, “FizzBang” if it is divisible by both three and five, otherwise it will output the number. Finally it will output the number of Fizzes and Bangs on two lines.  The output should be  FizzBang Demo  1  2  Fizz  4  Bang  Fizz  7  8  Fizz  Bang  11  Fizz  13  14  FizzBang  16  17  Fizz  19  Bang  6  4 *GNU Debugger – gdb* gdb can be used to analyse programmes created with both assembly language and c.  Use gdb on the c and assembly executables (one at a time!). Remember to set the interface to intel format with **set disassembly-flavor intel**  Set a break point at the first line of code. Run the code so it stops at the breakpoint.  Use the commands  *layout asm*  *layout regs*  to set up the screen to show you the code and the registers.  Step through the code and take not of where the values for the loop counter are stored and where the text is stored. You will need to be able to compare how your two programs use the registers. Report You need to submit a report of your comparison of your two programs.  It should give an overview of how each works – it need not be too detailed as you will have commented your code well..  It should describe how each program uses the registers and memory – which values are stored where and how they are accessed.  You should also include a comparison of how long each program took to run and how much memory it used (text, data, bss). You will find the linux commands *time* and *size* useful here. | |
| **FEEDBACK (how & when?)** | Feedback will be provided through Moodle by Monday 3rd February 2020 |

**Marking Guide**

Part One

|  |  |  |
| --- | --- | --- |
| **Component** | **Overall** | **Content** |
| **C Code** | | |
| Data Loads Correctly | 30% | 1. Data loads from file 2. To Binary Tree 3. In Correct Order |
| Node Diagram Produced | 30% | 1. Diagram produced 2. Correctly |
| Data Stored to File Correctly | 30% | 1. All data written to file 2. In the correct order |
| Coding Style | 10% | 1. Comments, variable names, structure, etc. |
| **C Document** | | |
| Document – Code Description | 30% | Describes the function of the code correctly and well |
| Document – Memory Description | 60% | Explains how the data is stored in memory correctly and clearly |
| Document Style | 10% | Document structured well, correct spelling & grammar, sentences, paragraphs, etc. |

Part Two

|  |  |  |
| --- | --- | --- |
| **Component** | **Overall** | **Content** |
| **Code** | | |
| C Program Compiles, Runs and generates correct output | 20% | 1. Loops correctly 2. FizzBang ifs correct 3. Output correct |
| Assembly Program – output | 20% | 1. Outputs in the correct format 2. No Segmentation Faults! |
| Assembly program – loop | 25% | 1. Has a correctly functioning loop |
| Assembly Program – FizzBang comparisons | 25% | 1. Correctly implements FizzBang comparisons |
| Coding Style – (both) | 10% | 1. Comments, variable names, structure, labels, etc. |
| **Document** | | |
| Code Description | 30% | Gives a good overview of how the two programs work |
| Memory Description | 30% | Explains how the data is stored in memory correctly and clearly for both programs, referring to registers, the stack, heap as appropriate |
| Comparison | 30% | A comparison of the similarities and differences between the two versions of the program – focusing on register use and memory use (text, data, bss). |
| Document Style | 10% | Document structured well, correct spelling & grammar, sentences, paragraphs, etc. |

University Generic Assessment Description