Introduction to C#

|  |  |  |
| --- | --- | --- |
| **Assessment Task Number:** Part 4 – Activities on Files | | |
| **Unit Code(s):** | | **Unit Title(s):** |
| ICTPRG430 | | Apply introductory object-oriented language skills |
| ICTPRG440 | | Apply introductory programming skills in different languages |
| **Items to Submit** | | **Evidence Criteria and Assessment Decision-Making Rules** |
| 1. | Alphabetize a File | Create a C# console application that   * meets the activity brief, * compiles without errors or warnings, * runs as a stand-alone application and * generates the expected output without errors or crashing. |
| 2. | Testing Document | A document that records at least 2 tests for each program written (task 1 and 2).  Tests record input, expected output, actual output, and whether the test passed.  For tests that fail, code is reviewed and re-tested until all tests pass. |
| **Submission Requirements:** | | |
| You will need to submit the following:   * A Release build of each application that can execute as a stand-alone program * Your complete Visual Studio project * Your testing document in MS Word or PDF format   Be sure to remove any temporary build folders (i.e., the Debug and Release folders). Only project files, source code files, and any resource files used should be included in your submission.  Package all files in a single compressed archive file (.zip, .7z, or .rar) | | |

Student’s own self-assessment of outcomes

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Competency | Description | Result |
| Task 1 |  | meets the activity brief |  |
| compiles without errors or warnings |  |
| runs as a stand-alone application |  |
| generates the expected output without errors or crashing |  |
| Task 2 |  | meets the activity brief |  |
| compiles without errors or warnings |  |
| runs as a stand-alone application |  |
| generates the expected output without errors or crashing |  |
| Task 3 | Write a testing document:  Create a document to record your testing process.  As you test your programs, write down the input you pass into the program, the expected output, the actual output, and whether or not the test passed. (If your tests fail, review and modify your code, then keep testing and reviewing until your tests pass).  Record at least 2 tests for each problem. | A document that records at least 2 tests for each program written (task 1 and 2). |  |
| Tests record input, expected output, actual output, and whether the test passed. |  |
| For tests that fail, code is reviewed and re-tested until all tests pass. |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target of test | Test input | Test expected output | Test actual output | Pass? |
| Task 1 test 1 |  |  |  |  |
| Task 1 test 2 |  |  |  |  |
| Task 2 test 1 |  |  |  |  |
| Task 2 test 2 |  |  |  |  |

1. Task 1
   1. Programming design documentation (Task 1)

The following narrative was created before I started writing code, to guide myself through the principles of the task. It was written as commentary within the program, and I have extracted it wholesale from there.

* 1. Screenshots of design iterations
     1. Narrative outline of major code components required
     2. Narrative understanding of what I need to create
     3. Console explanation of what the program is going to do (for my own benefit as I go)
     4. First test (source code)
     5. First test (results)
     6. Second test (source code)
     7. Second test (results)
     8. Standardise code
     9. Improve formulas
     10. Confirm success
     11. Simplify
     12. Re-format
     13. Experimentation
     14. Innovate