ICT159 Foundations of Programming Assignment 2

Note for ICT283 STUDENTS:

This wording in this assignment is directed to ict159 students. It is an actual assignment in ict159 which is a prerequisite to this ict283.

For example, there is no due date listed in LMS for ict283, but there is for ict159 students. There are no marks for ict283 students as it is a revision exercise that must be completed before ict283 students can access topic 3 onwards.

You will submit this completed document and source code.

To help you revise quickly, revision notes from ict159 along with the topic videos are provided. See links in LMS.

DUE: Please refer to LMS

If you are unsure of anything, you need to get clarification early. So read this document very carefully.

OBJECTIVES:

- Construct algorithms to solve problems using a combination of sequence, selection and iteration constructs.
- Implement such algorithms in a common programming language, ***C***.
- Read/write data to/from files.
- Use arrays of records (structs).
- Search arrays of records.
- Apply the methodology of top-down design to the construction of solutions and implement these solutions in a modular way.

PREREQUISITES:

- Assignment 1 has been attempted, even if not submitted.
- All lab exercises for modular programming, arrays, Files and Data Structures have been completed

Worth:
This assignment is worth 20% of the total assessment for the unit ICT159.

This is an individual assignment and must be completed by you alone. Any unauthorised collaboration/collusion may be subject to investigation and result in penalties if found to be in breach of University policy

General Submission Guidelines:

Failure to adhere to these guidelines will result in marks being lost or a fail grade being awarded.

- Your assignment **MUST** be submitted via LMS containing ALL sections.
- To be accepted and receive marks, your submission **must** meet the following criteria:
    - Create a folder named ICT159_StudentID_Surname_Assignment2 where StudentID and *Surname* are respectively your student ID and your surname.
    - Underneath the subfolder ICT159_StudentID_Surname_Assignment2,
        - Create a subfolder named Code, which will contain all the program files (.c, .h) you created, any data file that your program is using), and the executable file.
        - A single document according to the supplied template in .DOCX or .PDF format (ICT159_StudentID_SurnameAssignment2.docx or ICT15_StudentID_SurnameAssignment2.pdf).
        - Nothing else that is not asked for.
    - Once done, compress the folder ICT159_StudentID_Surname_Assignment2 to obtain a file named ICT159_StudentID_Surname_Assignment2.zip. Upload to LMS the compressed file ICT159_StudentID_Surname_Assignment2.zip. You must use zip, no other compression format will be accepted
    - **Once you have submitted, log out of LMS, then log back in and check that your assignment submission is there.**
- The non-source code parts of the assignment must be word processed and proof read: no hand written work will be accepted.
- **Students must use the assignment template provided in this document.** Using a different or altered structure provided may result in marks being deducted or work not being accepted due to assignment requirements not being met.
- All questions and sections within questions **must be submitted in order: marks may reduce otherwise.** The supplied template ensures this requirement is met.
- The assignment requires the submission of the different sections as specified below. Your final submission should be prepared as a single document containing each of the sections in order. You can do this by copying and pasting your algorithms/code into this document. Code (when discussing code aspects as needed) and algorithms within this document should be formatted with a monospaced font such as Courier New to enhance their readability. Source code files are submitted separately. Source code is submitted as C compilable source code with the appropriate code file extensions. C compilable means that the GNU C compiler should compile your source. C++ compiler must not be needed.
- Algorithms that look like the code was written first and then word processed to look like an algorithm would receive no marks.
- Assignments must be received by the deadline. Late assignments will be penalised 10% per day late or part thereof.
- Assignments will not be accepted where there are more than 5 days late.
- Should you anticipate difficulty in completing this work by the specified deadline, you should contact the Unit Coordinator **via e-mail.**
- Extensions will not normally be granted very close to the due date or without suitable grounds. Also, other than in exceptional circumstances, the amount of extra

time that can be granted will normally be quite limited. If you have been granted an extension, you must include a printed copy of the e-mail confirming the extension with your assignment submission.

- You should keep a copy of all your work.

IMPORTANT NOTE:

Non modular programs will receive a 0 for the entire assignment even if they run correctly and produce the correct result. In other words,

- if you dump all the code in the main function, you will get a zero for the entire assignment
- Although you use multiple functions, but you put all the functions inside one .c file (the one that contains the main program) then you get a 0 for the entire assignment.

Modularity refers not just at the function/procedure level but at the file level as well.

Assignment Sections:

Your assignment must contain the following components either in the documentation or as separate electronic files, where indicated:

1. All assumptions made other than those stated in the question that you make about the problem. There will virtually always be assumptions you are implicitly making so think about this very carefully. Also be careful that you do not put in unnecessary assumptions. **(5%)**
2. Structure chart for your program. Show parameter passing. (**5%**)
3. Your algorithm written in a uniform fashion using a pseudocode or a similar style and adhering to the conventions required in the unit. Your algorithm should be presented at an appropriate level of detail sufficient to be easily implemented. Submit your high- level algorithm along with algorithms of your decompositions (refinements) as appropriate to the question. Algorithms that look like the code was written first and then word processed to look like an algorithm would receive no marks. **(20%)**
4. A set of test data in tabular form with expected results and desk check results from your algorithm. Each test data must be justified – reason for selecting that data. No marks will be awarded unless justification for each test data is provided. **(10%)**
5. Source code files (.c and .h) must be submitted and the source code must build (compile and link) to create an executable that operates correctly. Make sure you use the code style required in the unit. No marks awarded if the source code does not build and run. **(50%)**
6. Results of applying your test data to your final program (tabular form), including a sample printout of your program in operation. **(5%)**
7. Self-assessment, which should include **(5%)**
   - How successful you were in achieving the requirements and a discussion of any problems you encountered (2.5%).
   - In point form, a summary of what works and what does not work in your program. A false claim here would mean that marks for this component would

not be awarded. So make sure that you have tested your program thoroughly (2.5%).

Please submit the sections in order as shown in the assignment template in this document. Marks may be lost otherwise.

Question

This question extends Assignment 1. There is meant to be code reuse from assignment 1 if your design and code from Assignment 1 adheres to the principles of high cohesion and low coupling.

You will know if these principles were adhered to if you are able to reuse most of the modules from Assignment 1.

Before attempting this question,

- ***please complete the prerequisites*** *listed on the first page of this document.*
- ***Make sure that you have corrected your solution to Assignment 1 following the solution discussed during the lectures (please refer to the recorded videos)***

Assignment 2 will require Array of records (structs) with file I/O is needed.

In this assignment, instead of asking the user to input the amount from the keyboard, this time, the program will read its data from a text file named *coins.txt*. There can be 0 and up to 10 input lines, each line is formatted like the example below:

Jane 30 cents in AU$
Joe 85 cents in EUR
Jane 25 cents in AU$
Jane 15 cents in US$
…

Names are one-word strings. The same name can be repeated in the data file but the coin values/currencies can be the same or different. If the name is the same, your program must assume it relates to the same individual.

You are asked to write a program that reads the data from this file. For each individual, the program then must add up the coin amounts that belong to the same currency to obtain a total amount for that individual in that currency. It then needs to compute the change to be given for that individual for each of the currencies that that individual owns.

For example, the example input file above indicates that Jane has 30 + 25 = 55 cents in AU$ and 15 cents in US$. The program then needs to calculate how many coins (hereinafter referred to as change) it should give to Jane for the amount it owns in AU$ and for the amount it owns in US$.

You do not covert currencies.

The coins we have are the same as Assignment 1:

1) US$, which has four types of coins of 50, 25, 10 and 1 cents
2) AU$, which has four types of coins of 50, 20, 10 and 5 cents
3) Euro, which has four types of coins of 20, 10, 5 and 1 cents

The program should aim to give as much of the higher valued coins as possible (same as Assignment 1). Note that the amounts in each line have values from 1 to 99. However, when you sum up the all the lines of an individual, the total for that individual can exceed 99. If the file contains a value that is not in that range, it should display a message saying that the data file has incorrect values and should display the line numbers that contain these incorrect values.

Once your program has read in the data from *coins.txt*, your program will close *coins.txt* first, and then show a console screen menu as illustrated below:

```
1. Enter name
2. Exit
```

The program will continue to show the menu and execute the menu options until "Exit" is selected by entering the value 2 at the menu prompt.

When the user enters the value 1 at the menu prompt, your program will ask for a name. As an example, if the user enters the name Jane, the program will output:

```
Customer:
Jane 55 cents in AU$

Change:
50 cents: 1
5  cents: 1

Jane 15 cents in US$

Change:
10 cents: 1
1  cents: 5
```

Change values of 0 should not be shown.

If the user enters a non-existent name (e.g., Donald) at menu Option 1, which would not be in the array of records, your program will print:

```
Name: Donald
Not found
```

After the output is provided for menu Option 1, the menu is redisplayed.

Once the user enters 2 to exit, your program must write the coin, the currency and change data in .csv format for all names present in the file coins.txt to another file called *change.csv*. After writing the data to the file your program must exit. In *change.csv*, the data lines for Jane will look as follows, with each value separated by a comma and with the line terminated by newline:

Jane, the change for 55 cents in AU$ is 1,0,0,1
Jane, the change for 15 cents in US$ is 0,0,1,5

So in the example output, Jane has

- 55 cents in AU$: one 50 cent coin and one 5 cent coin. There are no 20 or 10 cent coins.
- 15 cents in US$: one 10 cent coin and five 1 cent coins. There are no 50 or 25 cent coins.

The output data file *change.csv*, if an individual's name appears more than once inside *coins.txt*, their name should appear only once per currency inside *change.csv* (with the accumulated coin amount and change for that currency).

You need to provide a test plan to fully test your algorithm and program, and provide an input data file, *coins.txt*, that contains test data in the specified format for testing your program.

Your solution (program and algorithm) should be modular in nature. Use a high cohesion and low coupling design.

Submit a structure chart and a high-level algorithm with suitable decompositions (refinement) of each step (low-level algorithm).

Note that for this problem, the principle of code reuse is very important, and a significant number of marks are allocated to this. You should attempt to design your solution such that it consists of a relatively small number of functions that are as general in design as possible, and you should have functions/subroutines that can be reused (called repeatedly) in order to solve the majority of the problem.

If you find that you have developed a large number of functions (code modules/subroutines) where each perform some similar or overlapping task (or have a lot of code that is repeated in the functions) to other modules/subroutines, then attempt to analyse your design to generalise the logic so that you have just one general version of the module/subroutine.
*[ICT283 note: This ICT159 approach is one of the approaches used in '**Abstraction**". To be able to pass ict283, you need to apply the concept of abstraction to everything you do]*

Code re-use is related to cohesion. So, be mindful of the cohesion exhibited by the function (module). So, if you have a function (module) that is doing more than one task, then cohesion is low, and, you will need to redesign to have high cohesion (module does only one thing).
*[ICT283 note: The above concept that you learnt in ict159 is known as the **Single Responsibility** Principle and is part of the principles known as **SOLID**. To be able to pass*

*ict283, you need to apply **SOLID** to your design of Data Structures and applications using these Data Structures].*

When submitting the assignment, delete this and all previous pages.

Assignment submission template is on the next page.

ICT159 Assignment 2

<Student Name>

<Student Number>

1. Assumptions (5%)

All assumptions made other than those stated in the question that you make about the problem. There will virtually always be assumptions you are implicitly making so think about this very carefully. Also be careful that you do not put in unnecessary assumptions. Assumptions like "The user understands English" or "The user has fingers to type" may appear to make sense but are pointless from the point of view of the program operation. So do not record such assumptions. ***(5%)***

- First assumption
- Second assumption
- …

2. Structure Chart (5%)

Structure chart for your program. Show parameter passing.

3. Algorithm (20%)

Your algorithm written in a uniform fashion using a pseudocode or a similar style and adhering to the conventions required in the unit. Your algorithm should be presented at an appropriate level of detail sufficient to be easily implemented. Submit your high- level algorithm (where necessary) along with algorithms of your decompositions as appropriate to the question.

Algorithms that look like the code was written first and then word processed to look like an algorithm would receive no marks.

<write algorithm here>
….

4. Test Table (10%)

A set of test data in tabular form with expected results and desk check results from your algorithm. Each test data must be justified – reason for selecting that data. No marks will be awarded unless justification for each test data is provided.

Add rows to the following table as needed. Table can span more than one page. Each test id tests only one condition for the desk check.

For this assignment, there can be up to 10 records in a data file. In the test table below, you might have one test id for 10 records. So the actual 10 records must be in one cell of the test table in the column *Actual data*. Of course there are other test conditions and you need to include those too.

Faking the outcome of any test will result in no marks given for this entire section. What that means is that if you have a few hundred tests which are fine, but you faked/falsified the outcome of just one, you will get a mark of 0.

Test id	**Test description/justification – what is the test for and why this particular test.**	**Actual data for this test**	**Expected output**	**Actual desk check result when desk check is carried out**	**Desk check outcome – Pass/Fail**
1					
2					
3					
4					

5. Code (50%)

Name and purpose of functions/modules in the source code files. Do not put actual source code here. Code exists as separate source code files that are submitted. Source code files (.c and .h) must be submitted separately and the source code must build (compile and link) to create an executable that operates correctly. Make sure you use the code style required in the unit. No marks awarded if the source code does not build and run.

Extend the following table as needed. Functions/modules need to match what is in the structure chart. If it is the same file name for a number of functions/modules, you write the file name once in the *File name* column for the first function/module listed in the table.

File name	**Name of Functions/modules in the file**	**Purpose of the Function/module**

6. Results of Program Testing (5%)

Results of applying your test data to your final program (tabular form), including a sample printout of your program in operation.

Add rows to the following table as needed. Table can span more than one page.

Each test id tests only one situation for the test run of the program. Table is copy/paste of the desk check with actual output column showing results of the program output. There should be no duplicated reasons listed in the second column.

Faking the outcome of any test will result in no marks given for this entire section. What that means is that if you have a few hundred tests which are fine, but you faked/falsified the outcome of just one, you will get a mark of 0.

Test id	**Test description/justification – what is the test for and why this particular test.**	**Actual data for this test**	**Expected output**	**Actual program output when test is carried out**	**Test run outcome – Pass/Fail**
1					
2					
3					
4					

After the above test table, copy/paste sample printouts of your program in operation. You can screen capture and paste here. Make sure you label each printout with the correct *Test id*.

7. Self-Assessment (5%)

Self-assessment of how successful you were in achieving the requirements and a discussion of any problems you encountered.

Write your self-assessment here. Use as much space as needed. Describe how well your solution meets the requirements. Explain how you can improve your solution. Discuss problems you encountered and how you resolved them.

Submit a separate file called *evaluation.txt*. This file has two headings, and you enter the required summary as dot points under the headings. The first heading is "**What works**" and the second heading is "**What does not work**". Do not make any false claims as marks for this component may not be awarded. Testing should be thorough.

The file *evaluation.txt* will also declare if you have checked each submitted file for viruses or malware. Name the tool and version number of the tool that you used to conduct the check. If the checks for viruses/malware are not made and the declaration is not shown in *evaluation.txt*, this assignment will not be marked, and no marks will be given to you. Any delay that results from virus or malware will incur the specified daily penalty for the assignment. Advice on how to do a malware scan is under Unit Info or Essential Resources at the LMS site for this unit.