**ICT159 Foundations of Programming**

**Assignment 1**

**DUE :** Please refer to LMS

Read this document very carefully a number of times. Do this as soon as it becomes available. Re-read it until you are sure of what is expected. If you need to clarify anything, do so as early as possible.

**OBJECTIVES:**

* Construct algorithms to solve problems using a combination of sequence, selection and iteration constructs.
* Implement such algorithms in a common programming language, ***C***.
* Apply the methodology of top-down design (studied in Week 6) to the construction of solutions and implement these solutions in a modular way.
* This assignment also serves as preparation for Assignment 2 and so this assignment must be completed.

**Worth:**

This assignment is worth 10% of the total assessment for the unit.

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 outlined in this document filled in.
* To be accepted and receive marks, your submission **must** meet the following criteria:
  + All source code files must be zipped into ICT159\_StudentID\_Surname\_Assignment1.zip
  + Executable file named must be Assignment1.exe
  + A single document using the Assignment Submission Template provided on pages 6-10. The template is in Word format; you must save it (and submit it) either in Word or in PDF format (eg: (ICT159\_StudentID\_Surname\_Assignment1.docx or \_StudentID\_Surname\_Assignment1.pdf), where *Surname* is your name). This constitutes the assignment documentation.
  + Summary evaluation file with file name evaluation.txt
  + Nothing else that is not asked for.
  + The documentation (i.e. non-source code parts) of the assignment must be Word processed and proof read before saving to PDF. No hand written work will be accepted (including diagrams).
  + **Students must use the Assignment Submission Template provided in this document (pages 6-10).** Using a different or altered structure than that 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 be reduced otherwise.** The supplied template ensures that you do not forget to meet this requirement.
  + The assignment requires the submission of the different sections as specified on the next page.
  + Your final documentation submission should be prepared as a single document containing each of the sections in order. All algorithms and any source code (for example, when discussing code aspects) included in this document should be formatted with a monospaced font such as Courier New to enhance their readability. Source code files are submitted separately as a zip file, along with a summary evaluation file *evaluation.txt*. Source code is submitted as C compilable source code with the appropriate code file extension.
  + 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 submitted to LMS by the deadline (AWST). Late assignments will be penalised 10% per day late (including weekends) of the mark achieved for the assessment.
  + Assignments will not be accepted where submission is more than 5 days late.
  + Where circumstances make it impossible to submit the assignment by the due date and time, you must contact the Unit Coordinator (via email) immediately, **and definitely prior to the submission date and time**. Extensions **will not** be granted if your request is made **after the due date and time**, and the above late penalty will be applied if your submission is late.
    - An extension will only be granted for legitimate (verifiable) reasons. Outside work commitments, heavy study load, other assignments being due at the same time, falling behind in your unit work, your computer crashed, you lost your usb, etc., do **not** constitute legitimate reasons. All students may have multiple assignments due around the same time or heavy work commitments; you need to effectively manage your time. Also, you should *always* keep backup copies of your work in case something goes wrong with your computer, usb, etc. *As a Murdoch student, you have access to onedrive cloud storage when you login to* [*https://account.microsoft.com/*](https://account.microsoft.com/) *using your Murdoch credentials. You are expected to use cloud storage as your backup for all work related to your studies.*
    - The Unit Coordinator will decide if an extension should apply, and if so, the duration of the extension. If an extension is granted a copy of the unit coordinator's response email **must** be included at the beginning of your submitted documentation, otherwise the late penalty may be applied.

You should keep a copy of all your work. Keep the backup of your work in cloud storage like OneDrive.

**Assignment Sections:**

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

1. All assumptions made about the problem and your solution (e.g., data types, currency, language, etc.). There will virtually always be assumptions that you are implicitly making so think about this carefully. However, your assumptions cannot contradict the question requirements. Also be careful that you do not put in unnecessary assumptions (e.g., things that are stated in the question). **(3%)**
2. Structure chart for your solution. This demonstrates the design of your solution. The structure chart must show data flow (i.e. parameter passing and return values). **(10%)**
3. Your algorithm written in a uniform fashion using a pseudo-code and adhering to the conventions required in the unit. Your algorithm also demonstrates the design of your solution and must be consistent with your structure chart. The algorithm should be presented at an appropriate level of detail sufficient to be easily implemented. Submit your high-level algorithm and the low-level decompositions of your solution, 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. **(15%)**
4. A set of test data in tabular form with *expected results and desk check results* from testing 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. **(15%)**
5. Source code file (.c) and program executable must be submitted (separate to your documentation). The source code must be consistent with your algorithm (both high- and low-levels). The table in section 5 must be completed. 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. **(40%)**
6. You are free to use Windows, Mac or Linux to compile your program. Please indicate in your documentation on which platform you have compiled your code.
7. Results obtained by applying your test data to your final program. That is, you must provide sample printouts of your program in operation (i.e. actual program output from applying the test values nominated in the test table in point 4 above). **(10%)**
8. Self-assessment of how successful you were in achieving the requirements and a discussion of any problems you encountered. **(7%)**

The self-assessment must be submitted in two parts:

1. A self-assessment write up done here in the document template.
2. You need to also submit a separate file named *evaluation.txt*. This file will have, 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. 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.

***Points 1-4 and 6-7 above relate directly to corresponding sections in the assignment submission template in this document. Please submit the printed sections in order as shown in the template. Marks may be lost otherwise.***

**Assignment Question:**

*You should read this question as if the change giver is an artificial intelligence-based bank teller who has to give amounts of money to customers by automatically working out the amounts.*

You are asked to write a modular solution (algorithm and C program) that will accept as input an amount of money in a certain currency and return the optimal number of coins in that currency.

The amount of money should be an integer value in the range of 1 to 95 inclusive.

The program should first ask the user to select a currency among one of the following three:

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

(Note: these coins are randomly chosen just for the purpose of this exercise.)

It should then ask the user to input an amount of money (a number between 1 to 95 inclusive). Your program should then return the minimum number of coins (in the selected currency) that sum up into that number.

Your solution should also ensure that, if the chosen currency is the AU$, the input value should be in multiples of 5.

Based on valid input, your solution should calculate how many coins of each denomination should be returned, and display this to the user.

The solution should aim to give as much of the higher valued coins as possible. For example, a poor solution for an input of 30 cents would give six 5 cent coins. A correct solution would give a 20 cent coin and a 10 cent coin.

After each output, the user should be asked whether they wish to continue (i.e., enter another amount) or exit/terminate the program.

Your solution (algorithm and program) should be designed using a modular approach. This requires the submission of a structure chart, a high-level algorithm, and subsequent decompositions of each step (i.e. low-level algorithms).

Note that structure chart is different from flowchart. Structure chart is about the modular decomposition of the problem; please refer to the lecture slides of Week 6 (Modular Programming).

Note that for this problem, the principles of code reuse and high cohesion are particularly important and a significant number of marks are allocated to these aspects of your design.

* You should attempt to design your solution such that it consists of a relatively small number of modules, but still meets the modular design best practice requirements of this unit. In particular, strive to have one module that can be reused (called repeatedly) to solve the coin calculation problem. If you find that you have developed a large number of modules where each performs a similar task, or that you have a lot of instructions that are repeated in one module, then attempt to analyse your design to generalise the logic so that you have just one general version of the module.
* Be mindful of the cohesion exhibited by each module. If you have a module that is doing more than one task (i.e. demonstrating low cohesion), then you should re-design it to have high cohesion.

**Please note: your solution must be modular in design; a maximum of half marks for the design components (i.e. structure chart, algorithmic design, program design and implementation) will be allocated if this is not adhered to.**

**Also note: your solution must perform user input validation to ensure that only correct values are able to be entered.**

**Important notes:**

* When your program asks the user to enter a numerical value, the user might type in characters. Normally this is an invalid input. However, in this assignment, you are not required to check for this type of invalid inputs.

**You should use the Assignment Submission Template below (pages 6-10) for your documentation submission component.**

**Fill in the appropriate sections as you develop your assumptions, structure chart, algorithm, test strategy, test results, and your evaluation.**

**When you are ready to submit the assignment, you can copy this document and delete this and all previous pages.**

**ICT159 Assignment 1**

***<Student Name>***

***<Student Number>***

1. **Assumptions**

*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. However, your assumptions cannot contradict the assignment question. Also be careful that you do not put in unnecessary assumptions.* ***(3%)***

* First assumption
* Second assumption
* …

1. **Structure Chart**

*Structure chart for your program. Show parameter passing. (10****%****)*

1. **Algorithm**

*Your algorithm written in a uniform fashion using a pseudo-code 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.* ***(15%)***

<write algorithm here>

….

1. **Test Table**

*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.* ***(15%)***

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. There should be no duplicated reasons listed in the second column.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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** | **Desk check outcome – Pass/Fail** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |

1. **Code**

*Name and purpose of functions 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, .h) must be submitted separately in a zip file, and the source code must build (compile and link) to create an executable that operates correctly.*

*In the column, “Purpose of the function/subroutine”, you will need to ensure that the function/subroutine that you have written does only one thing: Single responsibility principle.* Find out what is meant by the “Single responsibility principle” or it relates to the cohesion principle.

High level functions like *main*() call other subroutine/functions to get the job done. These other subroutines/functions call lower level routines (designed using a structure chart) to do single tasks. As an example, a function call *sum*, only does adding items together. This function will not ask for user input and will not do any output of the result to screen or file.

*In the column “Single responsibility? (Y/N)” you need to ensure that the code implementation of the function/subroutine does only one thing (see footnote [1]), in which case the answer is Y for yes. If you find that the code function/subroutine is doing more than one thing, you must refactor the function/subroutine. Do that by first examining your algorithm and structure chart to determine how to further modularise your design.  
  
Make sure you use the code style required in the unit. No marks awarded if the table below is not filled in or if the source code does not build and run.* ***(40%)***

Extend the rows in the following table as needed. Functions need to match what is in the structure chart and algorithm. If there are a number of functions in the same file, you write the file name once in the *File name* column for the first function listed in the table. Successive functions (in the same file) do not need to list the file name repeatedly.

|  |  |  |  |
| --- | --- | --- | --- |
| **File name** | **Name of function/subroutine in file** | **Purpose of the function/subroutine** | **Single responsibility? (Y/N)** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. **Results of Program Testing**

*Results of applying your test data to your final program (tabular form), including a sample printout of your program in operation.* ***(10%)***

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. This table is a copy/paste of the desk check, except the actual output column shows the results of the actual program output. There should be no duplicated reasons listed in the second column.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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** | **Program test outcome – Pass/Fail** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |

After completing the above test table, copy/paste printouts of your program in operation. Use the values (column 3) in your table to do the test runs. You can screen capture and paste below. Make sure you label each printout with the correct *Test id*.

1. **Self Assessment (7%)**

*Self-assessment of how successful you were in achieving the requirements and a discussion of any problems you encountered. This write up is done in this document in the space provided below.*

Write the self-assessment here. Use as much space as needed.

**You also need to 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. *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.*

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.