**ICT283 Revision Exercise**

***James Carppe***

***34827095***

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%)***

* The ‘coins.txt’ file’s lines will always be formatted as “[NAME] [AMOUNT] cents in [CURRENCY]”.
  + [NAME] is a string (char \*) no longer than 99 characters.
  + [AMOUNT] is an int that has a value less than 99 – values over 99 will throw a warning and be rounded down.
  + [CURRENCY] is one of three values:
    - US$
    - AU$
    - EUR
* The ‘coins.txt’ file is always present in the executable’s directory.
* The computer has adequate memory to read and compute on the ‘coins.txt’ file’s customers.
* The computer has the correct software to run the executable.

1. **Structure Chart (5%)**

*Structure chart for your program. Show parameter passing.*

1. **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.*

Function main

Call: Load Customers & Store the result as customers

Call: Show Menu

End Function

Fucntion Load Customers

Pass in: File name

Open the file & verify it opened successfully

Loop over all the lines in the opened files:

Create a Customer object with the lines data

Add the new Customer object to an array of Customers

Create a Customers object and set it to store the array of Customers and set an int field to the number of customers

Pass out: Customers object

End Function

Function Show Menu

Pass in: Customers object

Loop:

Display the menu

Read the option selected from the user

Break the loop if the exit option is selected

Prompt the user for a name

Read the name from the user

Display the selected Customer if it exists

End Function

1. **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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 | Check change is calculated correctly | AU$: 25 | 20c x 1  05c x 1 | 20c x 1  05c x 1 | Pass |
| 2 | Check name search works | Desired customer: Joe  Joe 55 cents in US$  Jam 20 cents in AU$ | Joe 55 cents in US$  Change:  50c x 1  01c x 5 | Joe 55 cents in US$  Change:  50c x 1  01c x 5 | Pass |

1. **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.*

|  |  |  |
| --- | --- | --- |
| **File name** | **Name of Functions/modules in the file** | **Purpose of the Function/module** |
| calculator.c | debug\_Change | Debug function used to print the values of a Change struct |
| free\_Change | Free the memory allocated when a Change struct is created |
| calculate\_change | Calculates the change for an amount and a given currency |
| currency.c | currency\_from\_string | Converts a string input into an int representing a currency |
| string\_from\_currency | Converts an int and writes a string to a provided char\*\* pointer |
| currency\_type\_from\_int | Converts an int to a type from a Currency enum |
| customer.c | validate\_amount | Validates an input amount |
| validate\_amount\_limit | Validates an input amount including an upper bound check |
| debug\_Customer | Debug function used to print the values of a Customer struct |
| debug\_Customer\_Cluster | Debug function used to print the values of a Customer\_Cluster struct |
| free\_Customer | Free the memory allocated when a Customer struct is created |
| free\_Customer\_Cluster | Free the memory allocated when a Customer\_Cluster struct is created |
| create\_Customer | Creates a Customer struct and validates the fields |
| create\_Customer\_Cluster | Creates a Customer\_Cluster struct and validates the fields |
| customer\_add\_amount | Adds an amount to an existing Customer structs amount and validates the new value |
| customer\_compare | Compares two Customer structs and returns if they are equal |
| customer\_merge | Merges two Customer structs into one |
| files.c | get\_lines\_in\_file | Gets the number of lines in a FILE\* |
| open\_readonly\_file | Opens a file and returns the opened file inside a ReadOnly\_File struct |
| Open\_readwrite\_file | Opens a file and returns the opened file inside a ReadWrite\_File struct |

|  |  |  |
| --- | --- | --- |
| **File name** | **Name of Functions/modules in the file** | **Purpose of the Function/module** |
| interface.c | interface\_output\_customer | Outputs the data from a Customer struct in the format required |
| interface\_output\_change | Outputs the data from a Change struct in the format required |
| interface\_output\_complete\_customer | Uses the above two functions and outputs a complete customer |
| interface\_show\_menu | Prints the menu to the console |
| interface\_use\_menu | Shows the input & returns the chosen option |
| interface\_get\_name | Gets a name from the user |
| loader.c | process\_customer | Creates a customer based on information read from a line in coins.txt |
| load\_customers | Loads multiple customers from a coins.txt |
| math.c | math\_get\_closest\_divisble\_number | Gets the closest number to a base number, that is also divisible by another number |
| main.c | main | Program entry point |
| get\_customers | Loads customers from a coins.txt file for the program to use |
| use\_menu | Contains a loop that shows a menu and outputs a desired customer based on their name |

1. **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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 | Check change is calculated correctly | Jane 33 cents in AU$  Joe 85 cents in EUR  Jane 15 cents in US$  James 10 cents in EUR | Joe:  20c x 4  05c x 1 | -- -- --  <CUSTOMER>  Joe 85 cents in EUR  <CHANGE>  20 cents: 4  5 cents: 1  -- -- -- | Pass |
| 2 | Check name search works | James:  10c x 1 | -- -- --  <CUSTOMER>  James 10 cents in EUR  <CHANGE>  10 cents: 1  -- -- -- | Pass |