C++ project:

Bureau de change game

Student Name: Adam Wallbanks

Student ID: n0997892

Table of Contents

[Analysis: 2](#_Toc101621897)

[Aims: 2](#_Toc101621898)

[Problem identification: 2](#_Toc101621899)

[Research of existing problems: 3](#_Toc101621900)

[Program Requirements: 3](#_Toc101621901)

[Design and implementation 4](#_Toc101621902)

[Problem Decomposition: 4](#_Toc101621903)

[Testing: 6](#_Toc101621904)

[Evaluation: 8](#_Toc101621905)

[Possible Improvements: 9](#_Toc101621906)

# Analysis:

## Aims:

I would like to make a bureau de change game as it is not something I have seen made before. To do this I will make use of an API that will get the current exchange rate based off a base code sent. This game will use a UI to display questions and receive answers. This game will be very mathematical as it will require the user to multiply the currency rate by a random number to get the correct amount.

## Problem identification:

The main problems with this program will be:

* integrating the Currency API as it will make a request which will require using a random currency taken from a list of currencies.
* Using a graphical user interface to display the data and to receive user input using a textbox and a button.
* Rounding the numbers to 2 decimal places so the answer is in a simpler and more readable format and monetary values are to 2 decimal places.
* The biggest problem of all will be that I will be using external libraries that I have no prior experience with so this will cause a steep learning curve.

I also have to ensure that I stay within the working outlines for this programs purpose as the purpose of this program is to create a game that mimics a bureau de change employee who converts an amount from one currency to another but the difference with this game compared to a real-life employee is that the conversion is done manually by the person whereas in real life the conversion is done automatically on a computer program.

To ensure I stay within the working outlines I will continuously as the questions listed below throughout the development of this program:

* Does this fit into the concept of a real-life employee?
* Do the monetary values match up with the values in the real world?
* Is the data in a presentable format?
* Can someone who has not coded the game understand it and how to play?

## Research of existing problems:

After researching bureau de change games, I have found that there really aren’t any games at all so the best way to look at existing problems is to use real world examples such as the supermarket bureau de changes such as Tesco travel bank, Sainsburys travel money, john Lewis’s finance etc...

This does mean that there is not much to research in terms of a video game format, but I can research real world examples to make the game more realistic and to fit the concept of a real bureau de change employee.

Since there is a lack of programs to research. This will mean I will go in blind to any possible pre-existing problems or drawbacks so I will have to fix problems along the way as and when I find any. Although with an object-oriented approach it should be simpler to fix any problems and add any new features if they are needed.

# Program Requirements:

My program requirements contain the basic parts of the program that are required for the program to function at the lowest level will be considered critical as these must be met whereas other parts will be considered non-critical. Based on previous research on existing problems and real-world example I have produced this requirements specification for my program.

|  |  |  |
| --- | --- | --- |
| Requirement | Solution | Critical? |
| Calling the API with a custom query | I will need to use the external library Libcurl to do this and change the URL by making it data type char and using strcat\_s to add the query to the URL | yes |
| Receiving data from the API and storing it | Store the data given back as a string so it can be used later in the program | yes |
| Selecting a random currency | Using a generated random number as the position of the currency in the currency array | No as the program could still work without it being random |
| Finding it within the received API data string | Using string.find to find the position value of the currency in the string | yes |
| Selecting a random exchange value  Storing the correct currency and rate in a sub string | Creating a float array and using a generated random number as the positional value  Using string.substr to get the currency and rate sub string using positional values of the start of the required currency and it will stop at the start of the next currency | No as number does not have to be random could just be pre-set.  yes |
| Calculate the answer from the rate and the exchange value | Convert both the rate to a float using stof () and then just multiply them together | yes |
| Take users input and store it | Use of cout and cin to output the question and the input is the users’ answers | yes |
| Check if users input matches the calculated answer | If statement checking if user answer == to calculated answer and outputting correct | yes |
| Using a conditional loop dependant on lives | Make a while loop dependent on lives being greater than 0 and including the question and API call within the loop so the question is different each time | yes |
| Using a GUI to display the question, lives and to receive user input | Use of SFML graphics and TGUI widgets to display the relevant data and also the integration of images and textures | No as it can be work without GUI integration |
| Lives are decremented after wrong answer | This would be a part of the if statement checking if the user answer is correct as there would be an else statement that outputs incorrect and removes a life | yes |
| Game over outputted after lives run out | This would be after the while loop so that it will only output after the conditional loop breaks | yes |
| Using the GUI to make a game over window | Use of SFML graphics to make the window and sf text to display the text | No could just be outputted on the shell |
| Using a sound for the game over window | Using SFML audio to add sound | No game does not need sound to function |

# Design and implementation

## Problem Decomposition:

There are multiple problems that will have to be overcome for a successful outcome. The best way to do this is to break up larger problems into more manageable ones.

The first problems that has arisen is calling the API and receiving data and storing said data. This can be broken down into: determining the base currency to make the call with(the query) ,creating the base URL needed for a call, adding the query to the end of the URL, using a writecallback function sourced from [Save curl content result into a string in C++ - Stack Overflow](https://stackoverflow.com/questions/9786150/save-curl-content-result-into-a-string-in-c) to store the received data as a string.

Here is a flow chart showing the API call and storing the received data:

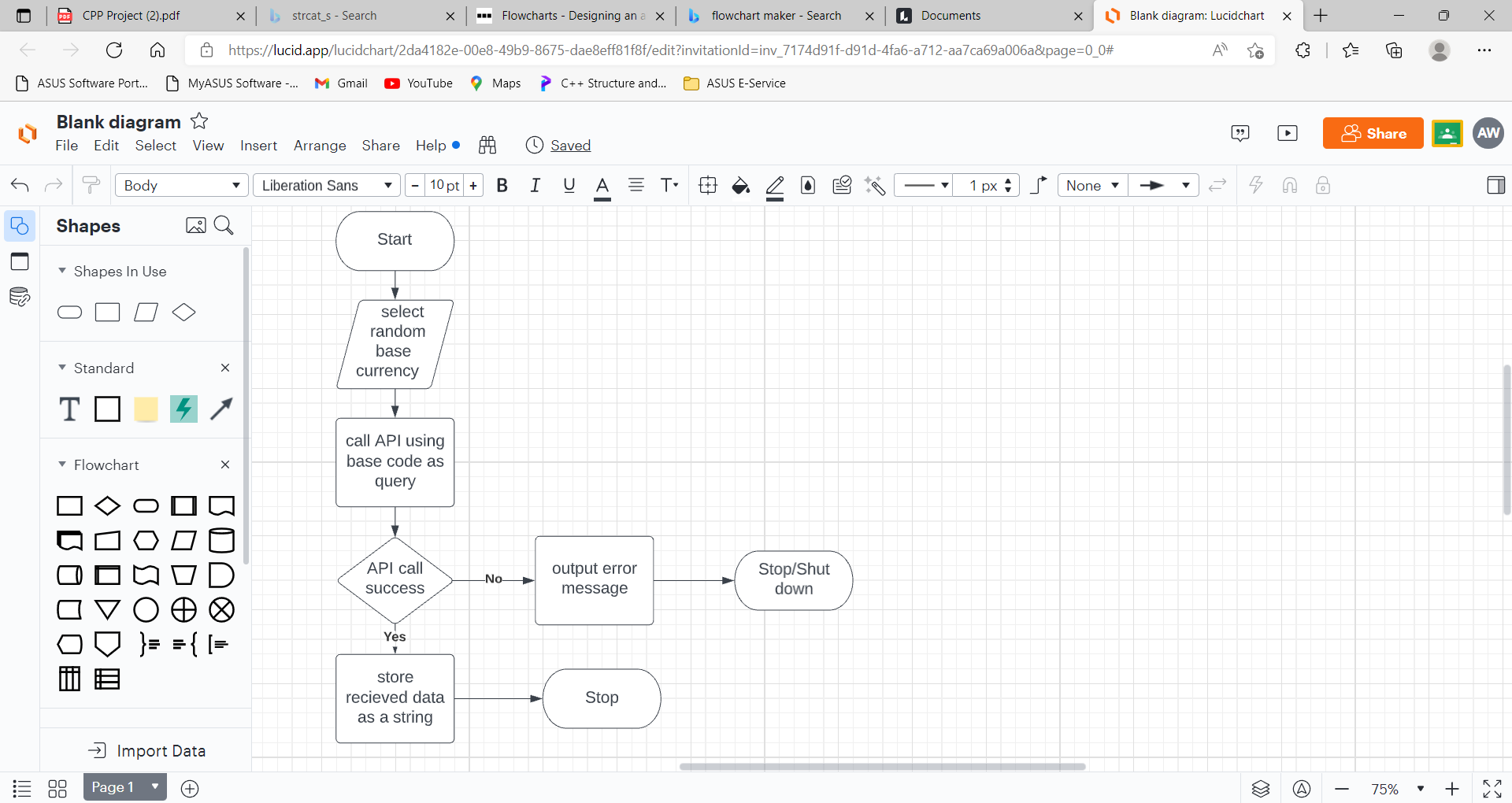


Figure 1: a flow chart to show how an API call is carried out within the program

Second problem is creating a question string. This can be broken down into: determining a random currency to be converted to, finding the currency and rate in the API data string, displaying the currency and rate, selecting a random exchange value, creating a question by including the exchange value, base currency and currency to be converted to, storing the question as a string

The third problem is displaying all the relevant data onto the GUI. This can be broken down into: creating a SFML render window, converting the SFML window to a TGUI window, creating an image sprite for the background, creating an image sprite for the desk, creating an image sprite for the customer, creating a SFML text variable for the question, creating a SFML text variable for the base code and rate, adding all the SFML objects to a TGUI SFML canvas, adding the SFML canvas’ to the GUI window, creating a label to display the lives left, adding the label to the GUI, displaying the GUI window.

The fourth problem is receiving user input using the GUI. This can be broken down into: creating a textbox to receive the users answer input, adding the textbox to the GUI, creating a button to submit the answer, creating a function that gets the text from the textbox on button press, closing the window upon button press so it can be redrawn to with a new question and an updated lives tally and then redisplay, adding the button to the GUI, displaying the updated GUI window.

The fifth and final major problem is outputting previous searches stored within a text file. This can be broken down into: creating a function to append each search to the text file, creating a function to read the text file and output each line within the file, calling the read and output text file when required and outputting the results onto the shell, this will be done upon user input to the question of whether they want to see previous searches.

Throughout the program it is very important to consider the design and the colours of everything hence why I have used an aqua background on the shell with white text colour as it gives more of a feel that you are playing a game and I have also incorporated a fitting background to make it more game like but still with a realistic feel as this is the kind of game I am going for where it is moderately realistic but it still feels like you are playing a game.

# Testing:

There are many tests that could be done to check the functionality and error handling of this program but due to the size allowed for this report I will only be covering ones I consider to be major ones.

The table of tests carried out can be seen below:

|  |  |  |
| --- | --- | --- |
| Test | Functional Requirement | Expected output |
| Inputting a negative number (e.g.-4) into the textbox | The program should be able to deal with incorrect inputs | Incorrect, correct answer displayed life deducted |
| Inputting a number not to 2dp into the textbox | The program should be able to deal with incorrect inputs | Incorrect, correct answer displayed life deducted |
| Inputting a character into the textbox | The program should be able to deal with incorrect inputs | Incorrect, correct answer displayed life deducted |
| The submit answer button enables the user input to be retrieved | The user input must be able to be received so it can be used to check against the correct answer | You answered user input string |
| Input incorrect answer | The program should be able to deal with incorrect inputs | Incorrect, correct answer displayed life deducted |
| Input correct answer | The program should be able to recognise correct inputs | Correct displayed and no life deducted |
| Random currency selected each time | The program should be able to generate a random number as a positional value for the array of currencies | Base code displayed and rate displayed both currencies different |
| The base code is displayed in a readable format | The program data should be readable and simple to understand | Base code displayed with no speech marks |
| The currency and rate are displayed in a readable format | The program data should be readable and simple to understand | Currency and rate displayed with no speech marks or commas |
| Input a number more than 2 dp | The program should be able to deal with incorrect inputs | Incorrect, correct answer displayed life deducted |
| Input y on the shell for previous search question | The program should be able to call and carry out a function when required | Previous searches displayed program continues after output |
| Input n on the shell for the previous search question | The program should be able to call and carry out a function when required | The program carries on without displaying the previous searches |
| Input something other than y or n for the previous search question | The program should be able to deal with incorrect inputs | The program should carry on without displaying previous searches |
| A random exchange value is selected each time | The program should be able to generate a random number as a positional value for the array of conversion numbers | A different exchange value / conversion number is displayed each time |
| The program loops until lives = 0 | The program should be able to loop with a condition | Multiple questions asked until said condition is met |
| The question is displayed in a readable format | The program data should be readable and simple to understand | The question string is displayed within the speech bubble on the image sprite of the woman |
| The API call is successful | The program should be able to make an API call from a query (base currency) | No error displayed |
| the data received from the API is stored as a string | The program should be able to receive and convert data from the API | Currencies and base codes are correctly displayed |

# Evaluation:

As stated in the table below all requirements that were listed in the table in the section named program requirements have been met.

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement | Solution | Critical? | Met? |
| Calling the API with a custom query | I will need to use the external library libcurl to do this and change the URL by making it data type char and using strcat\_s to add the query to the URL | yes | yes |
| Receiving data from the API and storing it | Store the data given back as a string so it can be used later in the program | yes | yes |
| Selecting a random currency | Using a generated random number as the position of the currency in the currency array | No as the program could still work without it being random | Yes, sometimes random number was the same, but a failsafe added to counter that |
| Finding it within the received API data string | Using string.find to find the position value of the currency in the string | yes | Yes |
| Selecting a random exchange value  Storing the correct currency and rate in a sub string | Creating a float array and using a generated random number as the positional value  Using string.substr to get the currency and rate sub string using positional values of the start of the required currency and it will stop at the start of the next currency | No as number does not have to be random could just be pre-set.  yes | Yes  yes |
| Calculate the answer from the rate and the exchange value | Convert both the rate to a float using stof () and then just multiply them together | yes | Yes |
| Take users input and store it | Use of cout and cin to output the question and the input is the users’ answers | yes | yes |
| Check if users input matches the calculated answer | If statement checking if user answer == to calculated answer and outputting correct | yes | Yes |
| Using a conditional loop dependant on lives | Make a while loop dependent on lives being greater than 0 and including the question and API call within the loop so the question is different each time | yes | yes |
| Using a GUI to display the question, lives and to receive user input | Use of SFML graphics and TGUI widgets to display the relevant data and also the integration of images and textures | No as it can be work without GUI integration | yes |
| Lives are decremented after wrong answer | This would be a part of the if statement checking if the user answer is correct as there would be an else statement that outputs incorrect and removes a life | yes | yes |
| Game over outputted after lives run out | This would be after the while loop so that it will only output after the conditional loop breaks | yes | yes |
| Using the GUI to make a game over window | Use of SFML graphics to make the window and sf text to display the text | No could just be outputted on the shell | yes |
| Using a sound for the game over window | Using SFML audio to add sound | No game does not need sound to function | yes |

## Possible Improvements:

Even though all of the requirements listed in program requirements were met this does not mean that the program is perfect and can’t be improved.

Some improvements that could be made are:

* Levels could be added to the game so it gives the player a greater sense of achievement and means users can progress. This could be done in the form of job rankings and maybe a salary increase.
* The questions could get gradually more difficult, so it is harder to last longer in the game
* A timer could be added for the amount of time for each question and the more questions answered the shorter the time to answer would be
* There could be different customers each time instead of just one customer for each conversion question
* The game could utilise text to speech, so the question is spoken instead of being written and displayed.
* There could be customisable colour backgrounds for people who are colour blind
* Etc….