FACTOID Q/A

421-AP1-AS sec.0721

Shayan Delbari – 2436670

Edward Angeles – 2431513

Brett Trudel – 2437549

2024-12-02

Table of Contents

Introduction

This project's main objective is to answer user provided factoid questions from a reference text using a C# console application. The project was designed to apply the tools and techniques that they have learned in the past semester to achieve this main objective. It has enriched our ability to collaborate, communicate, and better understand algorithms and rule-based systems in C#.

Problem Explanation

The first step to tackling any problem is to understand the task at hand. A factoid question is a closed-ended questions that starts with one of the following questions words*: who, when, where, how many,* or *how much*. The question will be asked by a user, and the program must do the following tasks (which correspond with the modules themselves):

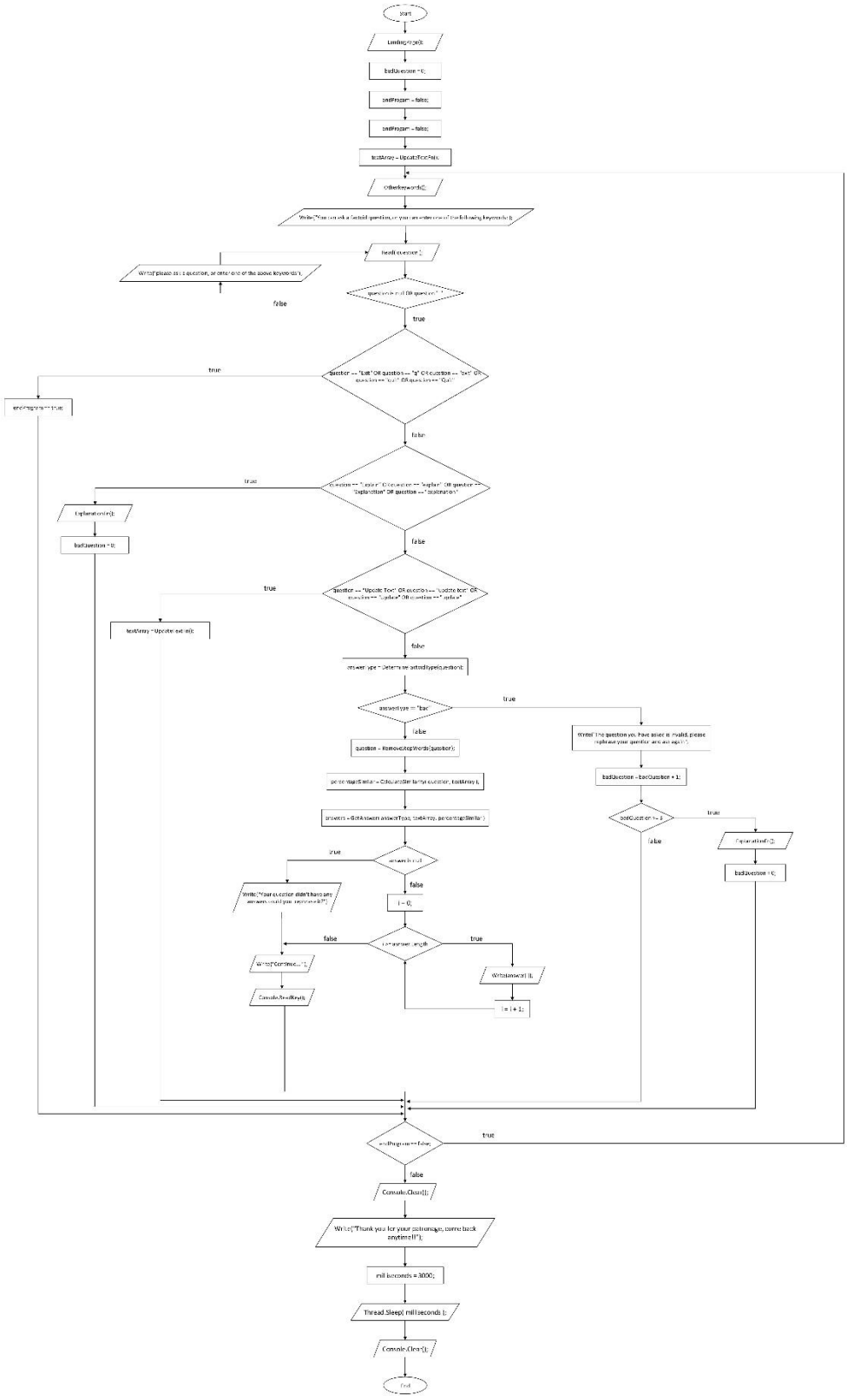
1) Analyzing the type of question
2) Retrieving the most relevant information from the reference text
3) Extracting the answer and displaying it to the user

As factoid questions are close-ended, they allow for the program to find a singular answer from the references text. This makes answering these questions simpler than their open-ended counterparts, but within the limitations of this project, mainly not being able to use dynamic datatypes, proved challenging. To tackle this, the team took a user focused approach, anytime the program was interacting with the user, and otherwise worked to make each function within each module work as smoothly together as possible.

Program Overview

From the get-go, the program was broken into three modules: Question Analysis, Calculating Similarity, and Answer Retrieval. The team broke it down into smaller functions to make it more manageable and see where they were going with each. For better legibility, they have chosen to clear the console at the start of any new question, or any time the user will be adding any input. See the below flowchart to visualize the overall architecture of the program.

Program Architecture



Landing Page

(Line 91 in C# program file)

As the user first launches the program, the first function to greet them is called the Landing Page, a very simple text displaying procedure that introduces them to the basics of the program, and what keyword operate other key aspect of the program.

Update Text Function

(141)

They are then required to update the reference text for the program. They can update the text instead of asking another question at any point by entering the keyword “Update”. The Update Text function has a couple other implemented functions: a loop to ensure the text is updated, and four functions that will edit the text to be better formatted for the other parts of the program.

A while loop that will continue to cycle until the text is not null or until it has any characters in the string.

Replace

(168)

This takes the word “Inc.” and replaces it with “inc” as the capital letter and period will come back as a problem in upcoming functions, such as Split.

Split

(478)

Divides the text from a paragraph to an array of sentences. It finds the punctuation and uses them as the place when to stop filling one element and start filling the next.

On issue the team ran into with this is when there is a decimal or comma in a number (i.e. $2.50 or 1,000,000). To get around this, they added an if statement to prevent the Split function from moving onto the next element if the char immediately after the period or comma,

Trim

(594)

Shortens the length of a string if the characters at the beginning or end are only spaces (‘ ’) by slicing the sentence.

To Lower

(562)

Changes the first letter of a sentence to a lower case. It uses two preset 1D arrays with the letters a-z (one array is all capital and the other is all lower). It checks to see if the first letter of the sentence is in the array of capital letters, and it will replace it with the corresponding lowercase letter.

One issue that the team ran into was if there was a person’s name that started the sentence. The Get Person function requires people’s names to start with a capital. They found, if the second word starts with a capital letter, the first word is most likely a person’s name, and therefore the To Lower function will not change the capital letter of the first word.

Q & A Loop (Main)

(7)

This loop is the main function of the program itself. After the console is cleared, it will remind the user of the keywords for other functionalities, using a simple procedure (called Other Keywords) that displays these like a menu. It then asks the user for a question. If no question is asked (the question asked has the value of null or “” (a string with no characters)) it will ask the user to ask a question until a value with at least on character is returned.

Using a series of if / else if statement, it will filter out several keywords that will launch the program’s Other Menu Options. Should the question not be one of the keywords, the program will assume that it is a question and begin to break it down to determine an answer. The first step it will go it is the Determine Factoid Type function.

In the Determine Factoid Type function if the question does not start with one of the factoid question words, it will return the answerType variable as “bad”. When this happens, the program, will ask the user to rephrase their question. If they ask three “bad” question, the

program will automatically take them to the Explanation function to explain again what types of questions they can ask. Before it resets the bad question counter to zero.

Providing the question asked was a good question. The program will put it through the Determine Factoid Type function which returns a string and is used for next in the Get Answer function.

To reduce a false high on the upcoming Calculate Similarity function, the program will then remove the *stop words* from the question. It uses the question (without stop words) and the reference text. The Calculate Similarity function outputs an array of floating-point numbers with an index that matches the index of the array of sentences of the text. Therefore, greatest number in the Percentage Similar Array will indicate the most similar sentence in the array of the text broken down into sentences.

Lastly the Get Answer function will use the Determine Factoid Type and the sentence with the most similarities to find the answer. Within the Get Answer function there are 4 sub-functions that correspond with the Factoid Types: Get Person, Get Place, Get Date / Time, Get Amount. The program will print the answer that is returned by the Get Answer function, then restart the loop until the user chooses to exit.

There are cases that the sub-functions in Get Answer are not able to return an answer, in which case they will return "null". In that case, the program will ask the user to rephrase the question and restart the loop.

Other Menu Options

Ending the Program

(26)

This will continue to loop until the user exits the application. This was achieved by running a Do/While loop that continues until a Boolean variable (endProgram) becomes true, which only happens if the user enters the keyword "Exit", "exit, or "q" into the question variable.

Upon breaking the Q & A loop, the program thanks the user, and after 3 seconds, closes itself.

Updating the Text

(141)

The user may update the text (see Update Text function above) to change the reference text without exiting the program. They only need to enter the keyword “Update”, “update”, “Update Text”, or “update text”.

Explanation

(186)

This procedure simply reminds the user what type of question they can ask to get an answer (i.e., *who*, *when*, *where*, *how many*, or *how much* questions). It does this by printing a couple lines.

Original Menu

The original plan for the menu was a switch / case menu that would have look like with the following options: 1 – Ask a Question, 2 – Update Text, 3 – Explain Factoid Questions, 4 – Exit.

The team opted to change the program to continuously ask for questions and only defer from that course if the user inputs one of the keywords, so they did not need to return to the menu every time the wanted to ask a question as this is the programs main functionality.

Determine Factoid Type

(217)

To determine the type of factoid question, one only needs to look at the first word:

First Word of Question	Expected Answer Type	Function Returns
Who	Person	“getPerson”
Where	Place	“getPlace”
When	Date or Time	“getDateTime”
How many	Numeric Amount	“getAmount”
How much	Numeric Amount	“getAmount”
Default (anything else)	Invalid / Not Factoid Question	“bad”

The exception to this rule is of course if the first word is “How” they need to know the second word as well. The team was able to work around this by isolating the first word by finding the first space (‘ ’) in the string. Then if the first word is how, the program will iterate past the space and continue adding letters until the second space.

It then run the first word through a Switch / Case to return the corresponding string. This string is used in the Get Answer function.

Pseudocode:

```
Function DetemineFacoidType(string question);
Var   sting firstWord, factoidType;
      boolean done;

BEGIN
i = 0;
done = false;
WHILE done == false;
      firstWord[i] = firstWord + question[i];

      IF firstWord[i] != ‘ ‘ AND firstWord != “How” THEN
            IF firstWord == “How” THEN
                  firstWord = firstWord + question[i];
            ELSE
            IF question[i] == ' ' AND firstWorld == "How" THEN
            firstWord = firstWord + question[i];
            i = i +1;
            firstWord = firstWord + question[i];
            END IF;
      END IF;
      i = i +1;
      IF question[i] == ‘ ‘ THEN
            done = true;
      END IF;
END WHILE;
```

```
SWITCH (firstWord) DO
      case “Who”:
            factoidType = getPerson;
            Break;
      case “When”:
            factoidType = getDateTime;
            Break;
      case “Where”:
            factoidType =    getPlace;
            Break;
      case “How many”:
            factoidType = getAmount;
            Break;
      case “How much”:
            factoidType = getAmount;
            Break;
      default:
            Write (“The question you have asked is invalid, please
            rephrase your question and ask again.”);
            Break;
END SWITCH;
Return factoidType;
END
```

Removing Stop Words – from Question

(255)

In the original assignment, this was meant to be done to the full text, but the team determined that the main purpose of this was to reduce the number of similar words between the question and the text that were not important (words like *the*, *a*, or *is* that would result in a false high when checking for similarity). They opted to remove these words from the question as it was shorter and would take less iteration and less time.

The Remove Stop Word function works by looking at each word in the question and comparing it to a programmer provided 1D array or common stop words. If the word is not a stop word it is added to a new string, and if it is a stop word, it is not. It continues this until it gets to the end of the question.

Calculate Similarity

(450)

This function takes the text (which is now an array of sentences) and the question (without stop words) and find how similar they are. To start it splits the question into an array of words using the Split function. It then loops through the elements of the text array, sentence by sentence. For each sentence, the loop will:

- Split the sentence into words using the Split function
- Loop through the words of the sentence's array of words
    - For each word of the sentence of the text, it will compare it to the words of the question. If they are a match, it will add one to a Similarity Counter
- Once each word of the sentence is checked for similarity, it will add the percentage of the sentence ((Similarity Counter / Length of Sentence) *100) that is similar into a results array. The index of this array matches the index of the array of sentences that makes up the text.

(For clarification see pseudocode below)

By the end, this function worked well, but a couple issues came up along the way. One the team faces was with the datatype. Because they decided to calculate the percentage by dividing an integer by and integer, the array kept coming back with all zeros. Once they

realized this was a limitation of the integer datatype, changing it to a floating-point (double) datatype to quickly resolve this issue.

Pseudocode:

```
Function static double[] CalculateSimilarity(string question, string[] text);

VAR   double[text.length] result;
      string[] questionWords, wordText;
      double similarityCounter;
      Split(string text, [object? character = null]);
      integer i, j, u;


BEGIN
questionWords[] = Split(question);
for i from 0 to text.length do
      similarityCounter = 0;
      wordtext[] = Split(text[i]);

      for j from 0 to word.length do
            for u from 0 to questionWords.length do
                        if (wordText[j] == questionWords[u]) then
                        similarityCoutner = similarityCounter +1;
                        break;
                        endif;
            endfor;
      endfor;
     result[i] = similarityCounter / questionWords.Length * 100;
endfor;
```

```
return result;

END
```

Get Answer

(622)

This function is called in the main loop after making sure that the question type is not "bad". This function gets the question type, the text as an array of sentences, and the result from the Calculate Similarity function which is an array of doubles, and it returns the answers as an array of strings. Before the main functionality, the program looks for the index of the maximum value in the similarity array which is done in a separate function to reduce redundancy. The main functionality is inside a for loop that runs three times because they decided that if there is in an answer that in a sentence not in the top three similar ones, it is probably irrelevant. Inside this loop based on the type of the questions the program calls the corresponding function while expecting a null answer. If the result is not null, it immediately returns the answer but if it is null, it sets the current high index to zero and in looks for the highest index again for the next round. If no result is found after the for loop is finished they return a null value which is handled in the main function.

Within all four of the sub-functions, there is a condition stating that: if the size of the answer is equal to zero, return null as the answer. This signifies that there was no answer found with in the sentence with the most similar sentence. This will check the next most similar sentence, for the same answer type using the same sub-functions. The team's thinking behind this: if the first and second most similar sentences don't return with an answer with and answer. The less similar sentences are more likely to return a wrong answer then a right one. If the two most similar sentences both return null, the program will ask the user to rephrase their question, and loop back to the beginning.

Pseudocode:

```
Function string[]? GetAnswer(string questionType, string[] text, double[] similarity);

VAR     string[]? result;
        integer maxIndex;
        integer sentenceTimes;
```

```
BEGIN
result = null;
maxIndex = HighestIndex(similarity);
sentenceTimes = 3;
FOR i FROM 1 TO sentenceTimes DO
        IF (questionType == "getPerson") THEN
                result = GetPerson(text[maxIndex]);
                IF (result != null) THEN
                return result;
                ELSE
                similarity[maxIndex] = 0;
                maxIndex = HighestIndex(similarity);
                ENF IF;
        END IF;
        IF (questionType == "getPlace") THEN
                result = GetLocation(text[maxIndex]);
                IF (result != null) THEN
                return result;
                ELSE
                        similarity[maxIndex] = 0;
                        maxIndex = HighestIndex(similarity);
                END IF;
        END IF;
        IF (questionType == "getDateTime") THEN
                result = GetDateTime(text[maxIndex]);
                IF (result != null) THEN
                        return result;
```

```
                ELSE
                similarity[maxIndex] = 0;
                maxIndex = HighestIndex(similarity);
                END IF;
        END IF;
        IF (questionType == "getAmount") THEN
                result = GetAmount(text[maxIndex]);
                IF (result != null) THEN
                return result;
                ELSE
                similarity[maxIndex] = 0;
                maxIndex = HighestIndex(similarity);
                END IF;
        END IF;
END FOR;
return result;
END
```

Highest Index

(697)

This simple function returns the index of the greatest value in an array of numbers. It does this by setting the value in the first index of the array (a[0]) as the max, then iterating through the array. If it comes across a value that is larger than the current greatest value, it replaces the index. The loop ends when the index is no longer less than length of the array, and it returns the index of the greatest element as an integer.

Get Person

(264)

Once the Get Answer function calls this function, it will be run on the sentence of the text with the highest similarity to the question. It is looking for any answer type that is a person's

name. In this exercise that is any word that starts with a capital letter, and the rest of the word is lower case. People's names can be one or more words long. It iterates through the string of the most similar sentence until it finds a word that starts with a capital. Once that is has located that word, it determines the length of the word and returns the word (which is the person's name) as the answer. If the next word in the sentence also starts with a capital, it is more likely than not part of the same name.

One issue that they ran into, what if the first word of the sentence is capitalized, like all sentences in English have? This made the first word of any sentence return as a person's name. To overcome this the team created To Lower function to change every first word of the sentence to have a lowercase letter to start. This presented a new issue. What if the first word of the sentence is the person's name and is now no longer capitalized? To combat this, the To Lower function was changed to not alter the first word of the sentence if the second word if also capitalized. This is not a perfect solution. Consider the following sentences: "Madonna was born…", "When John Smith visited the new world …". If the first word of the sentence is a mononym, or the there is a single non-name word before a name starting as the second word. The team deemed this fix to be a better option than the original problem, and an acceptable limitation.

Get Location

(328)

Once the Get Answer function calls this function, it works much like the previous Get Person function, but as the text is formatted so locations are in all caps (ie, MONTREAL), it does not have the same issue as Get Person. The function looks for words that have the first two letters capitalized and returns the whole word as an element of the answer array. It stops if the next word (after a space) is not capitalized, or if the word that is all capitalized is followed immediately by a comma.

Get Date / Time

(361)

Once the Get Answer function calls this function providing the sentence with the highest similarity, this function iterates through the sentence looking for a character that is a number. At this point, based on the index, it either checks for 4 and 7 characters ahead to be '-' or '/' or it checks if the next 4 characters are all number. If the first condition is true, it adds the next 10 characters to the answer and sets the index to the right place for the next round. If the second condition is true, it adds the next 4 characters as an answer and sets the index to the right place for next round. Most of the time the index is in place where both

conditions can be true in which the first conditions will run and if there is no result the second one will run after it.

They initially had some problems with the index being out of range if the program is trying to look for 7 characters ahead when there is not that many left. But by adding couple conditions to check the place of the index, they have caught the errors before it happens.

Get Amount

(403)

Once the Get Answer function calls this function providing the sentence with the highest similarity, this function splits the sentence into words and then looks for words that have a number either at the first or the second character. If the word starts with a number, the program looks for a special character like '%' or '$' at the end of the word. If the second character is a number, the program will look for the same special characters at the beginning of the word. The special characters are in a static array defined at the beginning of the function.

The function initially was working with two pointers for the beginning and the end of the answer. However, the problem with this approach was it would only extract the first instants of possible answers, so they pivot to how the function currently is.

Limitations of the Code

There are many limitations of the code. Overall, the team is proud of what they have been able to accomplish with their collective knowledge, and the rules put in place by the assignment. Below are a few more limitations not listed in the

In the sub-functions of Get Answer, the array that the results are recorded in are only 10 elements long. If there are more than 10 answers, there will be an error. This has not shown to be a problem yet because typically they only return 1–2-word answers.

The program is limited to working with text that has been formatted to the assignment specified parameters. It requires the date to be YYYYMMDD separated with ‘/’ or ‘-’, and locations need to be all capitalized, etc.

The code can be done using much less code by using dynamic data structures and pre-existing C# functions (such as: .Split, .Join, .ToLower, etc.). The team understands this is part of the assignment but would be remised if they did not mention this at least one more time.

Get Amount can only recognize numbers that are linked to a '$' or '%' symbol. They originally wrote it in a way that any number would be recognized by get amount, but typically it would pick up years and dates often. They tried to include an exclusion criterion to exclude anything in the YYYYMMDD format, and any integer between 1500 and 2024 to exclude the years of most modern history, hedging their bets on what the reference text would likely include. The team realizes that the code is limited to only include percentages and currencies that use the '$' symbol.

Teammate Contributions

C# Code	
Main() // Q&A Loop	Shayan & Brett
OtherKeywords ()	Brett
Landing Page()	Brett
UpdateTextFn()	Brett
ExplanationFn()	Brett
DetermineFactoid Type()	Shayan
RemoveStopWord()	Edward
GetPerson()	Shayan
GetLocation()	Shayan
GetDateTime()	Edward
GetAmount()	Brett
CalculateSimilarity()	Edward
Split()	Shayan
Replace()	Shayan
ToLower()	Shayan
Trim()	Shayan
GetAnswer()	Shayan
Highest Index()	Shayan

Flowcharts	
All	Edward

Pseudocode	
Calc. Similarity	Edward
Get Answer	Shayan
Determine Factoid Type	Brett

Report	
Get Answer	Shayan
Get Amount	Shayan
Get Date / Time	Shayan
Rest of Report	Brett

Code Debugging and Console Effects	
All	Shayan

GitHub Repository

https://github.com/shayandelbari/factoid-project

References

Console Text Art generated by: ASCII Art Archive (Injosoft ©2024)

https://www.asciiart.eu/text-to-ascii-art