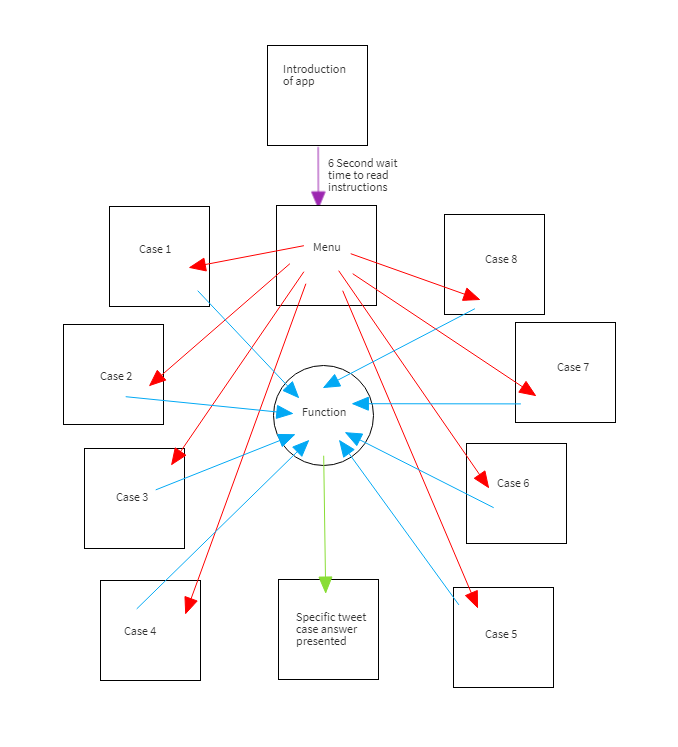
Max Korner’s Development Document

Abstract

For this project, we were asked to create an application that interrogates tweets from a certain documentation provided (sampleTweets.csv). This project was expected to count the number of tweets in a variety of ways as well as printing specific tweets as well choosing our own 4 unique tweets to pull. After finishing the project we were then expected to provide a GUI (Graphical user interface) which would be implemented into the project at a further date. For the GUI we used openframework.

Project Plan

|  |  |  |  |
| --- | --- | --- | --- |
| 23rd of April | 27th of April | 29th of April | May 1st |
| Begin Project | Finish Menu | Complete Code | Create GUI |
| Create basic structure (Set up source/menu/header) | Make sure menu is working correctly, with up and down keys working as intended including highlights | Finish off the code, make sure that the sample tweets is being previewed | Make sure GUI is finished |
| Update development doc | Update Development Doc | Update Development Doc | Finish Development Doc |



Evidence of Design

Wireframe

Technical Description

The application works as standard, simple source, header and main file. The main file consists of the standard packages to be included in the app, such as Iostream, fstream, string, windows.h and the Header file. The main int act’s as a control point on when each function should be presented with the introduction cout functions presented to the user. The introduction must come first as it gives the user’s an understanding on how to use the program while in terminal before the application is updated for the GUI. I was able to create a short delay using the Sleep() function which I used commonly throughout my code as it can provide a much cleaning experience for the user. Instead of having multiple lines of text which will group and make reading much harder. The system refreshes itself and provides the next input instead of keeping the original text on the terminal. Then the code moves onto the main menu, which is picked up using the header file from the source file. So for the design of the app I wanted users to have a much easier system and not have to actually data into the application which can confuse the app later on in the code when it’s come to inputting the wrong data.

To create the menu, 8 strings are made asking each a different question, the int pointer is the data for the menu, using the SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE) I am able to define the colours of the menu so that it is clear what setting is being highlighted by the user, I then used a while loop to create the key state for the user to input the settings, for the up and down keys. This system is very simple as it uses the amount of strings to go up and down so for other developers they can easily add different strings and change else if pointer line if (pointer == 8) to the amount they want to put to input different questions in the future. For the menu to work you must use the enter key on a highlighted function to create the data, once again using the else if function, I am able to create 8 different cases which highlight each question and build the anwer. For the first case I built the structure of each case with if (inFile.good()) {cout << "File being read..." << endl;while (!inFile.eof()). This line allows the file to be checked if it’s ready to be read, if the file is readable the terminal will output a setting saying the file will be read, once completed the next function of code will start, getline(inFile, line) answerTotal++ gathers the amount of lines in the sampletweet file and outputs the answer to answerTotal. The next cout function simply shows off the text to the terminal and the answer next to it. For the next function I introduced a new int “phrase” along with npos. Now this function was to gather the amount of data in the file on the specific question and add the number to the int. The int will then refresh it’s original asking value but keep the amount, they will then ask the same question but with capitals in different places as we are looking for words not letters so capitalisation can be different. I create the same phrase 3 different time asking different ways of capitalization for it to then push out the correct answer the phrase int. Then presenting the answer with a cout. Finally ended the code with a sleep(10000) to present the answer to the user before refreshing.

Test plan

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Did it work Y/N | Expected outcome | Outcome |
| Test 1: First output to the terminal with introduction | Y | Terminal shows introduction | First outcome the terminal presented the introduction but on the same line. Using the /n function I was able to create line spacing for the introduction to be more pleasing to the user. |
| Test 2: Include header and source file correctly without app not being built correctly | Y | Build Succeeded | Build succeeded without 0 failures |
| Test 3: Create Strings and highlighted menu | 1st time: N  2nd time: Y | Terminal shows menu and can correctly use up and down arrow keys to access different questions | When starting the coding, using the up and down arrow keys the menu would show but the down function wouldn’t work and it only highlighted when using the up function to go down. Made no sense. Then after realising that I forgot to close the } function to UP arrow code the code worked correctly. |
| Test 4: Correctly providing data to tweets | Y | Terminal shows the answers to the questions asked without no hiccups | The terminal correctly showed each answer from the sampletweet but when presented the code wouldn’t return to the menu which means that the user will have to restart the app to ask different questions each time. |
| Other environmental testing: Other windows machines and Macs | Y/N | App successfully builds on windows and macs machines. | The application will build itself correctly on a windows machine but when it comes to a mac machine the app wouldn’t work but I know this down to the use of the windows.h file so to be able to use the Mac in the future the menu will have to be built differently. |
| Does the GUI provide access to all functions used in the program |  | Yes | The gui provides a clear use for the user. |

Critical reflection

For this application I enjoyed designing the app, seeing the menu come to life and being able to provide the correct answers was quite satisfactory but I feel I let myself down when it came to asking the questions and user input. I haven’t allowed the user to input their own questions which I would like to build on in the future. As for the questions, I felt the question weren’t broad enough to fully use the app to it’s best extent as I have asked easy questions instead of testing myself to see if I can push the app to ask specific questions with user input. Though overall I am happy with the outcome of my application.

#include <iostream>

#include <string>

#include <fstream>

#include <windows.h>

#include "Header.h"

using namespace std;

//Creating menu value using int

int scrollingMenu()

{

string line;

int phrase;

ifstream inFile;

inFile.open("SampleTweets.csv"); //Opens File

string scrollingMenu[8] =//Name of the string scrolling data

{ "Count the number of tweets in the file", "Count the number of tweets that mention money", "Count the number of tweets relation to politics",

"Display all the tweets mentioning Paris", "Display all the tweets mentioning DreamWorks", "Display all the tweets mentioning Car", "Display all the tweets mentioning Garden",

"Display all the tweets mentioning House", "Display all the tweets mentioning Life", };

int pointer = 0;

int answerTotal;

while (true) { //Repeats the target statement if the conditions are true

system("cls"); //Clears output screen of programs

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 15);

cout << "Main Menu\n\n";

for (int i = 0; i < 10; ++i) //creating values for the asking options and exit

{

if (i == pointer) //if the user has selected of the following options

{

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 11); //highlight colour

cout << scrollingMenu[i] << endl;

}

else

{

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 15); //highlight colour

cout << scrollingMenu[i] << endl;

}

}

while (true) //This command will enable the up and down arrow keys, the if statement allows the option to move up from

{

if (GetAsyncKeyState(VK\_UP) != 0) // Allows the user to up arrow while navigating the menu

{

pointer -= 1;

if (pointer == -1)

{

pointer = 2;

}

break; //redraws new option highlighted

}

else if (GetAsyncKeyState(VK\_DOWN) != 0) //Allows the user to use the down arrow while navigating the menu

{

pointer += 1;

if (pointer == 8)

{

pointer = 0;

}

break;

}

else if (GetAsyncKeyState(VK\_RETURN) != 0) //Uses the enter key on different cases which are the strings above asking the questions to be inputted

{

switch (pointer)

{

case 0: //Case for number of tweets to be opened

if (inFile.good()) {

cout << "File being read..." << endl;

while (!inFile.eof()) {

getline(inFile, line);

answerTotal++; //Stores the number

//Provides the text including the stored number

} cout << "Total number of tweets is: " << answerTotal << endl;

Sleep(10000);

break;

}

case 1:

if (inFile.good()) {

cout << "File being read..." << endl;

while (getline(inFile, line)) {

if ((phrase = line.find("money", 0 /\*finds money text in sampleTweets/csv file\*/)) != string::npos || (phrase = line.find("Money", /\*Second time is for any capitlisation used in the code to provide certain result \*/ 0))

!= string::npos) { //Npos allows the static value to be it's greatest value for an element of type size

answerTotal++;

}

}

}

cout << "Total number of money mentioned tweets is: " << answerTotal << endl;

Sleep(10000);

break;

case 2:

if (inFile.good()) {

cout << "File being read..." << endl;

while (getline(inFile, line)) {

if ((phrase = line.find("Politics", 0)) != string::npos || ((phrase = line.find("Government", 0)) != string::npos || ((phrase = line.find("Brexit", 0)) != string::npos))) {

answerTotal++;

}

}

} cout << "Total number of political tweets is: " << answerTotal << endl;

Sleep(10000);

break;

case 3:

if (inFile.good()) {

cout << "File being read..." << endl;

while (getline(inFile, line)) {

if ((phrase = line.find("Paris", 0)) != string::npos || ((phrase = line.find("paris", 0)) != string::npos || ((phrase = line.find("PARIS", 0)) != string::npos))) {

answerTotal++;

}

cout << "Here are tweets relating to paris" << line << endl;

Sleep(10000);

}

} break;

case 4:

if (inFile.good()) {

cout << "File being read..." << endl;

while (getline(inFile, line)) {

if ((phrase = line.find("Dreamworks", 0)) != string::npos || ((phrase = line.find("dreamworks", 0)) != string::npos || ((phrase = line.find("DREAMWORK", 0)) != string::npos))) {

answerTotal++;

}

cout << "Here are tweets relating to Dreamworks" << line << endl;

Sleep(10000);

}

}break;

case 5:

if (inFile.good()) {

cout << "File being read..." << endl;

while (getline(inFile, line)) {

if ((phrase = line.find("Car", 0)) != string::npos || ((phrase = line.find("car", 0)) != string::npos || ((phrase = line.find("CAR", 0)) != string::npos))) {

answerTotal++;

}

cout << "Here are tweets relating to Car" << line << endl;

Sleep(10000);

}

}break;

case 6:

if (inFile.good()) {

cout << "File being read..." << endl;

while (getline(inFile, line)) {

if ((phrase = line.find("Garden", 0)) != string::npos || ((phrase = line.find("garden", 0)) != string::npos || ((phrase = line.find("GARDEN", 0)) != string::npos))) {

answerTotal++;

}

cout << "Here are tweets relating to Garden" << line << endl;

Sleep(10000);

}

}break;

case 7:

if (inFile.good()) {

cout << "File being read..." << endl;

while (getline(inFile, line)) {

if ((phrase = line.find("House", 0)) != string::npos || ((phrase = line.find("house", 0)) != string::npos || ((phrase = line.find("HOUSE", 0)) != string::npos))) {

answerTotal++;

}

cout << "Here are tweets relating to House" << line << endl;

Sleep(10000);

} break;

}

case 8:

if (inFile.good()) {

cout << "File being read..." << endl;

while (getline(inFile, line)) {

if ((phrase = line.find("Life", 0)) != string::npos || ((phrase = line.find("life", 0)) != string::npos || ((phrase = line.find("LIFE", 0)) != string::npos))) {

answerTotal++;

}

cout << "Here are tweets relating to Life" << line << endl;

Sleep(10000); //Forgot to add that this allows terminal to view the message to be read for 10000 miliseconds

}

}break;

}

}

}

}

return 0;

}