School of Applied Sciences And Technology

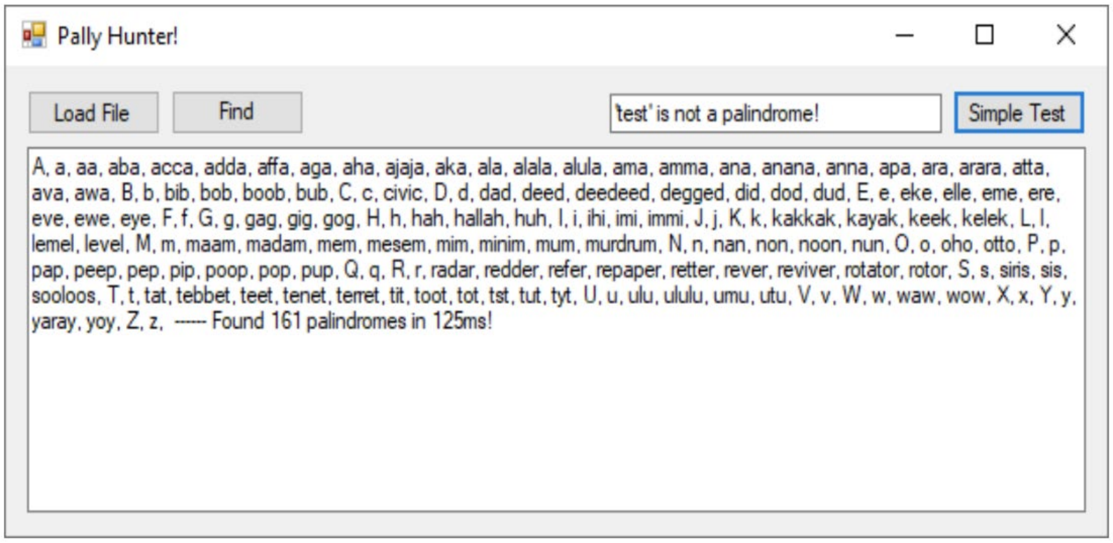
Department of IST

Program: CNT

CMPE1666 – ICA 16- Palindrome Hunter!

In this ICA you will use a thread to find palindromes (words that read the same backward as forward) in a collection of strings.

Create an application that has the following appearance:



```
'test' is not a palindrome!
```

```
A, a, aa, aba, acca, adda, affa, aga, aha, ajaja, aka, ala, alala, alula, ama, amma, ana, anana, anna, apa, ara, arara, atta,
ava, awa, B, b, bib, bob, boob, bub, C, c, civic, D, d, dad, deed, deedeed, degged, did, dod, dud, E, e, eke, elle, eme, ere,
eve, ewe, eye, F, f, G, g, gag, gig, gog, H, h, hah, hallah, huh, I, i, ihi, imi, immi, J, j, K, k, kakkak, kayak, keek, kelek, L, l,
lemel, level, M, m, maam, madam, mem, mesem, mim, minim, mum, murdrum, N, n, nan, non, noon, nun, O, o, oho, otto, P, p,
pap, peep, pep, pip, poop, pop, pup, Q, q, R, r, radar, redder, refer, repaper, retter, rever, reviver, rotator, rotor, S, s, siris, sis,
sooloos, T, t, tat, tebbet, teet, tenet, terret, tit, toot, tot, tst, tut, tyt, U, u, ulu, ululu, umu, utu, V, v, W, w, waw, wow, X, x, Y, y,
yaray, yoy, Z, z,  ------ Found 161 palindromes in 125ms!
```

Step 1

Since you are going to be looking for palindromes, you should create a separate method that can take a string and return a bool, indicating if the passed string is a palindrome. For full credit, the method should use recursion.

Use the top right controls initially to test your method. The user may enter a string, such as “test”. When the ‘Simple Test’ button is clicked, test the entered string with your method. Write back over the control with the entered string, in quotes, indicating if it is a palindrome or not (as shown). Test with various entered strings, validating that your method is returning the correct answer.

Step 2

A file has been supplied on Moodle that contains many, many words. You will load each line of the file into a List<string>. You may declare the List<string> at class level, as it will be used in different methods. Since you won’t have any strings until you load them, you may start this member out as *null*.

When the user clicks the 'Load File' button, use an OpenFileDialog to navigate to wherever you have downloaded this file. Read the entire contents of the file into your List<string>, creating the list as you do so:

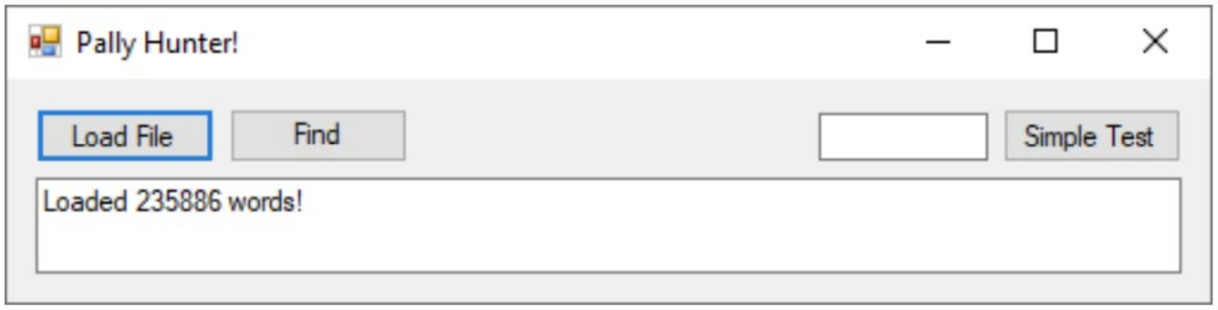
_lines = new List<string>(File.ReadAllLines(UI_OFD.FileName));

Note: ReadAllLines is a static method of the File type. The File type is available in the System.IO namespace. Note: ReadAllLines returns an array of string, but this may be converted to a List<string> through the List constructor as shown.

You should now have a List<string> that contains every line in the file

Note: We won't be doing any exception handling in this exercise, so make sure that you only select appropriate files for the load.

The bottom control in the form is a textbox that has been set to multiline. You may use this control after a load to show how many strings were loaded from the file, using the Count() method in List:



```
Loaded 235886 words!
```

Part 3

When the user clicks on the 'Find' button, you will start a **background** thread that will search the entire List<string> for palindromes using your palindrome testing method.

The thread body will create a new List<string> that will contain the found palindromes as you find them (you will add them to this list).

Once you have processed the entire source list, Invoke a method that will show the results, passing it the List of found palindromes. Display the palindromes with commas separating them, along with a message indicating how many were found, and how long it took to find them in ms (as shown in the first image).

Notes:

You should only have a List<string> to hold the loaded lines, a Thread to manage the list, and a System.Diagnostics.Stopwatch at class level. There should be no other fields at class level.

You may create a delegate type at class level for your callback (when processing is done). You will need the System.IO and System.Threading namespaces included for File and Thread.

The stopwatch should be started (restarted) when the 'Find' button is clicked and should be measured in the callback method when the processing is done

This application will require visual inspection of functionality and code.

Mark loss is at your instructor's discretion but will be applied consistently across all students.

Rubric: Total 30 marks

Item	Marks	Penalties
UI Design (9 marks)		
• UI is as directed	2	
• Use of OpenFileDialog is correct	3	
• Simple Test UI Correct.	2	
• Pally output as specified, with count and execution time.	2	
Code Design and Implementation (21 marks)		If multi-threading is not used for finding palindromes from the file, the whole assignment will be marked as 0.
• Palindrome-finding method uses recursion and working correctly (on simple test)	5	
• No extra fields added (data stored in appropriate controls)	2	
• File being opened and List built properly	3	
• Pally finding thread created runs as background thread.	2	
• Stopwatch used correctly for measuring execution time.	2	
• Separate method for testing Pally	2	
• Palindrome from file correctly found- count also correct	5	
Documentation		Penalty -1 to -6 for not meeting documentation requirements
• Code is well documented, where applicable		
• Code contains a programmer's block		
• Variable names are appropriate		
• Proper spacing between blocks of code		
• Control names are consistent and appropriate.		

Note: In this assignment, no need to give marks for the palindrome method or recursion. Focus on the thread and deduct marks for missing the basic tasks as well.