```
using System;
using System.Collections;
using System.Collections.Generic;
using Microsoft.SqlServer.Server;
using System.Data.SqlTypes;
using System.IO;
using System.Text;
using System.Text.RegularExpressions;

namespace CLRUDF
{
    public class SpeechParser
    {
        public static readonly string DEBUG_ROOT = "C:\\Projects\\CIS 612 Lab2\\";
        public static readonly string LOG_FILENAME = "ErrorLog.txt";
        public static readonly string TSV_FILENAME = "Speech_Inverted_Index.tsv";
        public static readonly string CONNECTION_STRING = "Data Source=DESKTOP-335I8BU;Initial Catalog=SPEECHES;Integrated Security=True";
        public static readonly string STOP_FILE = "stop.txt";
        public static readonly char[] JUNK_CHARS = {
            '`', '~', '!', '@', '#', '$', '%', '^', '&', '*',
            '(', ')', '_', '=', '+', '[', ']', '{', '}',
            '\\', '|', '\'', '\"', ';', ':', ',', '<', '.', '>', '/', '?'};

        public static List<Speech> GetSpeeches(string filename)
        {
            // This will be written in a very, very file specific manner, if the format of the file ever
            //  changes, this code WILL have to change with it
            string[] split = { "NEW ADDRESS=>" , "FIRSTNAME=", "LASTNAME=", "MONTH=", "DAY=", "YEAR=", "WEBLINK=",
                "FILELINK=", "SPEECH=" };
            List<Speech> speeches = new List<Speech>();
            try
            {
                string line = null;
                StreamReader reader = new StreamReader(filename);
                while (!reader.EndOfStream)
                {
                    line = reader.ReadLine();
                    string[] parts = line.Split(split, StringSplitOptions.None);
                    int day = -1, year = -1;
                    for (int i = 1; i < parts.Length; i += 9)
                    {
                        // Give a default value to day/year if they are bad
                        if (!int.TryParse(parts[i + 4], out day))
                        {
                            day = -1;
                        }
                        if (!int.TryParse(parts[i + 5], out year))
                        {
                            year = -1;
                        }
                        // Skip the first slot, its added by split and contains nothing
                        speeches.Add( new Speech
                        {
                            Firstname = parts[i + 1],
                            Lastname = parts[i + 2],
                            Month = parts[i + 3],
                            Day = day,
                            Year = year,
                            Weblink = parts[i + 6],
                            Filelink = parts[i + 7],
                            Text = parts[i + 8]
                        });
                    }
                }
                reader.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
            return speeches;
        }

        public static Dictionary<string, int> CreateSpeechIndex(List<Speech> speeches)
        {
            // Stores the number of times each term comes up across all speeches
            Dictionary<string, int> termFrequency = new Dictionary<string, int>();

            // Read in the stop words
            HashSet<string> stopWords = new HashSet<string>();
            try
            {
                StreamReader reader = new StreamReader(DEBUG_ROOT + STOP_FILE);
                while (!reader.EndOfStream)
                {
                    stopWords.Add(reader.ReadLine());
                }
```

```
                    reader.Close();
                }
                catch (Exception ex)
                {
                    throw ex;
                }

                // holds the current speech
                StringBuilder sb = new StringBuilder();
                string currentSpeech = "";

                // Iterate through each speech
                foreach (var speech in speeches)
                {
                    // Remove characters we do not care about
                    foreach (var c in speech.Text.ToLower())
                    {
                        if (!Array.Exists(JUNK_CHARS, j => j == c))
                        {
                            sb.Append(c);
                        }
                    }

                    currentSpeech = sb.ToString();

                    // Remove the stop words
                    foreach (var word in stopWords)
                    {
                        // Not the best way to do this
                        currentSpeech = currentSpeech.SafeReplace(word,  "", true);
                    }

                    // Remove the excess spaces in the string
                    RegexOptions options = RegexOptions.None;
                    Regex regex = new Regex("[ ]{2,}", options);
                    currentSpeech = regex.Replace(currentSpeech,  " ");

                    // Split the string into the query space
                    string[] words = currentSpeech.Split(' ');

                    // Update the index
                    foreach (var word in words)
                    {
                        // update the term frequency
                        if (termFrequency.ContainsKey(word))
                        {
                            termFrequency[word]++;
                        }
                        else
                        {
                            termFrequency[word] = 1;
                        }

                        // Update the document frequency
                        speech.UpdateFrequency(word);
                    }

                    // Clear out the current speech
                    currentSpeech = "";
                    // Clear out the string builder
                    sb.Clear();
                }

                return termFrequency;
            }

            [SqlFunction(FillRowMethodName = "FillRow")]
            public static IEnumerable InitMethod(string filename)
            {
                // Attempt to read in the speeches from file
                List<Speech> speeches = GetSpeeches(filename);
                // Hold the inverted index
                List<Tuple<string, int>> termFrequency = new List<Tuple<string, int>>();

                try
                {
                    // CreateSpeechIndex returns the term frequency
                    // Document frequency can be found as an attribute of the Speech class and will be
                    //  up-to-date when CreateSpeechIndex() exits
                    foreach (var term in CreateSpeechIndex(speeches))
                    {
                        termFrequency.Add(new Tuple<string, int>(term.Key, term.Value));
                    }

                    // TODO: Write out the speeches to the database
                    /*foreach (var speech in speeches)
                    {

                    }*/

                    // Write the term frequency out to a tsv file
                    StreamWriter output = new StreamWriter(File.OpenWrite(DEBUG_ROOT + TSV_FILENAME));
                    foreach (var term in termFrequency)
                    {
                        output.WriteLine($"{term.Item1}\t{term.Item2}");
                    }
                    output.Flush();
                    output.Close();
                }
                catch (Exception ex)
                {
                    StreamWriter log = new StreamWriter(File.OpenWrite(DEBUG_ROOT + LOG_FILENAME));
                    log.WriteLine(ex.ToString());
                    log.Flush();
                    log.Close();
                }

                // Return the speeches to the database, so it can handle updating the database
                return termFrequency;
            }

            public static void FillRow(Object obj, out SqlString term, out SqlInt32 frequency)
            {
                Tuple<string, int> pair = obj as Tuple<string, int>;
                term = new SqlString(pair.Item1);
                frequency =  new SqlInt32(pair.Item2);
            }
        }
    }
```