Separately assume there is a `1-by-T` row-vector of nonnegative numbers named `TaskWeights`. For notational purposes, denote the `TaskWeights` vector as

$$\begin{bmatrix} w_1 & w_2 & \cdots & w_T \end{bmatrix}$$

Using this notation, the *weighted task score* is defined as

$$WeightedTaskScore := \sum_{i=1}^{T} w_i s_i$$

Write a function named `weightScores` with the function-declaration line

```
function [GradeBookWeighted, B90] = weightScores(GradeBook, TaskWeights)
```

The function `weightScores` should use a `for` loop to add the field `WeightedTaskScore` to `GradeBook`, and for each element in `GradeBook`, sets its value to be the *weighted task score*, computed by combining the `ArrayTaskScores` with the `TaskWeights` as defined. The resulting struct array should be assigned to the variable `GradeBookWeighted`. Remember that adding a field to any one element of a struct array adds that field to all elements (i.e. all elements in a struct array have the same fields).

`weightScores` should also assign to the variable `B90` a `1-by-B` cell array of the names of the students whose WeightedTaskScore is greater than 90. This should take approximately 3 lines of code (a `for` loop should not be used for this task).

In your published file, create a small example (with, say, $N = 5$ and $T = 3$) and illustrate the functionality of `weightScores`.

Creating a Word Index
In the following problems you will write code that builds an index for text documents. This index will be a `struct` array where each element corresponds to a unique word in a group of documents. In each element of the struct array the word is stored in the `Word` field, the document numbers that the word is contained is in the `Documents` field, and the locations of the word in each document is in the `Location` field.

2. First, write a function named `InitializeIndex.m` with the following code:

```
function Index = InitializeIndex()
c10 = cell(1,0);
Index = struct('Word', c10, 'Documents', c10, 'Locations', c10)
```

This function has no input arguments and one output argument, an empty `1-by-0` struct array with the following fields:

- `Word`
- `Documents`

- Locations

Elements to this struct array fields will be appended with the InsertDoc function that you will create in the next problem.

3. Now, write a function named InsertDoc with the function-declaration line

begin code
```
function Index = InsertDoc(Index, newDoc, DocNum)
```
end code

The three input arguments are:

- Index: a 1-by-N struct array with the fields:
  - Word, where Index(k).Word is a char array(single row containing a word)
  - Documents, where Index(k).Documents is a 1-by-? double that contains the document numbers of documents that within which Index(k).Word appears,
  - Locations, where Index(k).Locations is a 1-by-? cell whose m-th element is a double row array that contains the locations of the appearance of Index(k).Word in the document specified by Index(k).Documents(m).
- newDoc: a 1-by-W cell array of words that represents a document to be added to the index; and
- DocNum: an integer identifying newDoc.

The process for inserting the data in newDoc into the Index is as follows: Loop through all of the words in newDoc,

- If a word is not one of Index(1).Word, ... , Index(N).Word, then a new element should be added to Index with the word(with all elements made lowercase) assigned to the Word field, the document number DocNum assigned to the Documents field, and a 1-by-1 cell array whose contents is a scalar double whose value is the location of the word in newDoc assigned to the Locations field. Comparison of words should be case-insensitive, so if 'matlab' is in the index then 'Matlab' or 'MATLAB' should not be added as a new element of the index.
- If a word is already one of Index(1).Word, ... , Index(N).Word and the document identified by DocNum is already included in the Documents field for this word, then the location of the word in newDoc should be appended to the corresponding array in the contents of the cell array in the Locations field.
- If a word is already one of Index(1).Word, ... , Index(N).Word, but the document identified by DocNum is not already included in the Documents field for this word, then the DocNum should be appended to the array in the Documents field, and the index of the word in newDoc should be appended to the contents of the appropriate cell array in the Locations field.

The proper behavior of this function can be illustrated with an example. Let's create an index assigned to the variable E7 and add the three "documents" shown below to the index.

```
Doc1 = {'Matlab', 'is', 'awesome'};
Doc2 = {'Programming', 'is', 'very', 'very', 'fun'};
Doc3 = {'I', 'love', 'Matlab'};
```

```
E7 = InitializeIndex;
E7 = InsertDoc(E7, Doc1, 1);
E7 = InsertDoc(E7, Doc2, 2);
E7 = InsertDoc(E7, Doc3, 3);
```

After running the code above E7 will be a 1-by-8 struct array, since there are 8 unique words in Doc1, Doc2, and Doc3. The first element of E7 is shown below. As can be seen the char array 'matlab' is the first element of the document corresponding to DocNum = 1 and the third element of the document corresponding to DocNum = 3.

```
         Word: 'matlab'
    Documents: [1 3]
    Locations: {[1]  [3]}
```

Similarly, the second element of E7 is:

```
         Word: 'is'
    Documents: [1 2]
    Locations: {[2]  [2]}
```

The fifth element of E7 is:

```
         Word: 'very'
    Documents: [2]
    Locations: {[3 4]}
```

Note that for the element E7(5) as shown above, the Locations field is a 1-by-1 cell array containing a 1-by-2 array, since the char array 'very' appears only in the document with DocNum = 2, but it appears twice in this document (the third and fourth elements).

Hints:

(a) My version of insertDoc has 19 lines, including the function-declaration line. Yours may be a bit longer, but there is no need for this to be a long function.

(b) Consider using the Matlab functions strcmp or strcmpi to perform comparison of char arrays. For more information type help strcmpi in the command window.

(c) Review all of the *comma-separated-list* syntax introduced in the module. Since `Index` is a `1-by-N` struct array, the expression {`Index.Word`} is equivalent to { `Index(1).Word, Index(2).Word, ... , Index(N).Word`}, and is therefore a `1-by-N` cell array containing all of the unique words current in `Index`

(d) For each word in a document under consideration, there are 3 cases to consider:

i. the word is not in Index (and must be appended properly)

ii. the word is in Index, but Document is not yet listed (in other words, the 1st occurance, in the document, of a word that is already in Index)

iii. the word is in Index, and Document is listed (the 2nd, or later, occurance, in the document, of a word that is already in Index)

Following that reasoning, here is some pseudo-code

```
begin code
function Index = InsertDoc(Index, newDoc, DocNum)
Given: Index, and a new document to insert into the Index
Get a cell-array of all words in Index
Loop through all words in the newDoc
   If the word is not in Index (or Index is empty)
      Add the word to Index (append at end, in .Word field)
      Correctly set the .Documents and .Locations fields
   Else  (word is already in Index)
      If this is the 1st occurance of this word in newDoc
          Add this DocNum to the correct location in .Documents
          Add the word location number into .Locations
      Else (2nd, or later, occurance of this word in this newDoc)
          Append the word location into .Locations
      End
   End
End
end code
```

Make sure that your `InsertDoc` function works properly for the example above. **Check every element of the `1-by-8` struct array `E7` to ensure it is correct!**

Now load the `3842-by-1` struct array `SongLyrics` contained in the `SongLyrics.mat` file. This struct has two fields:

- `Song`: a char array of the song name;
- `Lyrics`: a cell array containing char arrays of each word in the song.

For each value of `k`, from 1 to 3842, the value of `SongLyrics(k).Lyrics` represents a document as described above. All of these documents are lyrics from popular music. For the purposes of **building** the index, you will not use the values in the `Song` field.

Initialize an empty index, `BigLyricIndex`, and then use a `for` loop insert all of the documents in `SongLyrics` into the index. The resulting index should be a `1-by-19001 struct` array. **Hint:** Before doing the large `for` loop, try (manually, calling `insertDoc` a few times) inserting the first few documents, and examing `BigLyricIndex` to see that it's being constructed properly,

```
BigLyricIndex = InitializeIndex;
BigLyricIndex = InsertDoc(BigLyricIndex, SongLyrics(1).Lyrics, 1);
% examine BigLyricIndex
BigLyricIndex = InsertDoc(BigLyricIndex, SongLyrics(2).Lyrics, 2);
% examine BigLyricIndex
BigLyricIndex = InsertDoc(BigLyricIndex, SongLyrics(3).Lyrics, 3);
% examine BigLyricIndex
```

4. Now that you have created functions to initialize and build the index, we want to be able to search for documents that contain certain words or certain groups of words.

   Write a function named `FindDocsWithWords` with function-declaration line

   ```
   function matchingDocs = FindDocsWithWords(Index, words)
   ```

   The two input arguments are:

   - `Index`: a `1-by-N` struct array with the fields: `Word`, `Documents`, and `Locations`; and
   - `words`: a char array of a word to search for (e.g. `'yellow'`), or a cell array containing multiple char arrays of the words to search for (e.g. {`'submarine'`, `'the'`, `'walrus'`}).

   This function should return to the output argument, `matchingDocs`, an L-by-1 array that contains the `DocNum` of each document that contains all of the char arrays in the input argument, `words`. If no document in the index contains all of the char arrays in `words` then the output argument, `matchingDocs`, should be empty, namely 0-by-1.

   Hint: Use the `Matlab` function `intersect` in your `FindDocsWithWords` function. There is a quiz question in the online module titled "More Cell Operations" about using `intersect` with chars and cell arrays of chars that you should review.

   You should first test your `FindDocsWithWords` function on a small index, like that given produced from the example in the previous problem. Once you are confident that your `FindDocsWithWords` function is working correctly you should try it on the larger index that you created from `SongLyrics` with many different words.

   Some things to try:

   - how many songs have the word `'I'`?
   - how many songs have the word `'tonight'`?
   - how many songs have the word `'computer'`?
   - how many songs have the words `'I'`, `'love'` and `'you'`?
   - how many songs have the words `'I'`, `'am'`, `'the'`, and `'walrus'`?

   How can you use the returned Document Numbers along with `Index` to get the song titles of the matching songs?