Solution to Question No. 2 Part B:

B 2.1 Plagiarism rule and threshold:

The rules and threshold of plagiarism in my world are as follows:

1. If there are some important words in a document that are found to be copied in the same manner then the document is plagiarized.
2. If the document is one-to-one copied then the program will output 100% plagiarized.
3. If there are some important words in a document that are found to be copied in the same manner then the document is plagiarized
4. The threshold can be defined by the user, by default the program will calculate how much of the content in the document is plagiarized, or could be plagiarized.

B 2.2 Pseudocode for checking plagiarized content:

bag_words:

```
1.      Begin
2.      Create a HashMap mymap
3.      Read the data from the file and store it in char* data
4.      Declare variable tok and initilialize it with strtok(data, ".\n\t")
5.      While (tok ≠ 0)
5.1       Convert every character of tok to lower alphabet, remove the trailing and leading spaces
          and store it in to_add
5.2       If (to_add≠ '\n' && to_add ≠ '\0')
5.2.1       If (mymap contains to_add) then increment the value of key to_add in mymap
5.2.2       Else add to_add to hashmap and assign 1.0 as the value to the key to_add
5.3       tok = strtok(0, ".\n\t ")
6.      return mymap
7.      End normalize_words:
1.      Begin
2.      Declare total_words <- 0
3.      for (j = 0; j < length of bag ; j++)
3.1         total_words = total_words + bag -> data[j].value
4.      for (j=0; j < length of bag ; j++)
4.1         set the value for key bag -> data[j].key as current value for the key
          / total_words
5.      End
```

compute_tf_idf:

```
1. Begin
2. for (i = 0 ; i < length of bag ; i++)
    1.1 curr_word = bag - > data[i].key
    1.2 tf = bag -> data[i].value
    1.3 number_of_documents_with_word = 0
    1.4 for (j = 0 ; j < length of bags ; j++)
        1.4.1 curr_bag = bags -> data[j]
        1.4.2 if (curr_bag contains curr_word) then number_of_documents_with_word++
    1.5 idf = 1 + log (length of bags / number_of_documents_with_word)
    1.6 set the bag with key curr_word as tf * idf
3. End
```

4. **compute_cosine_similarity:**

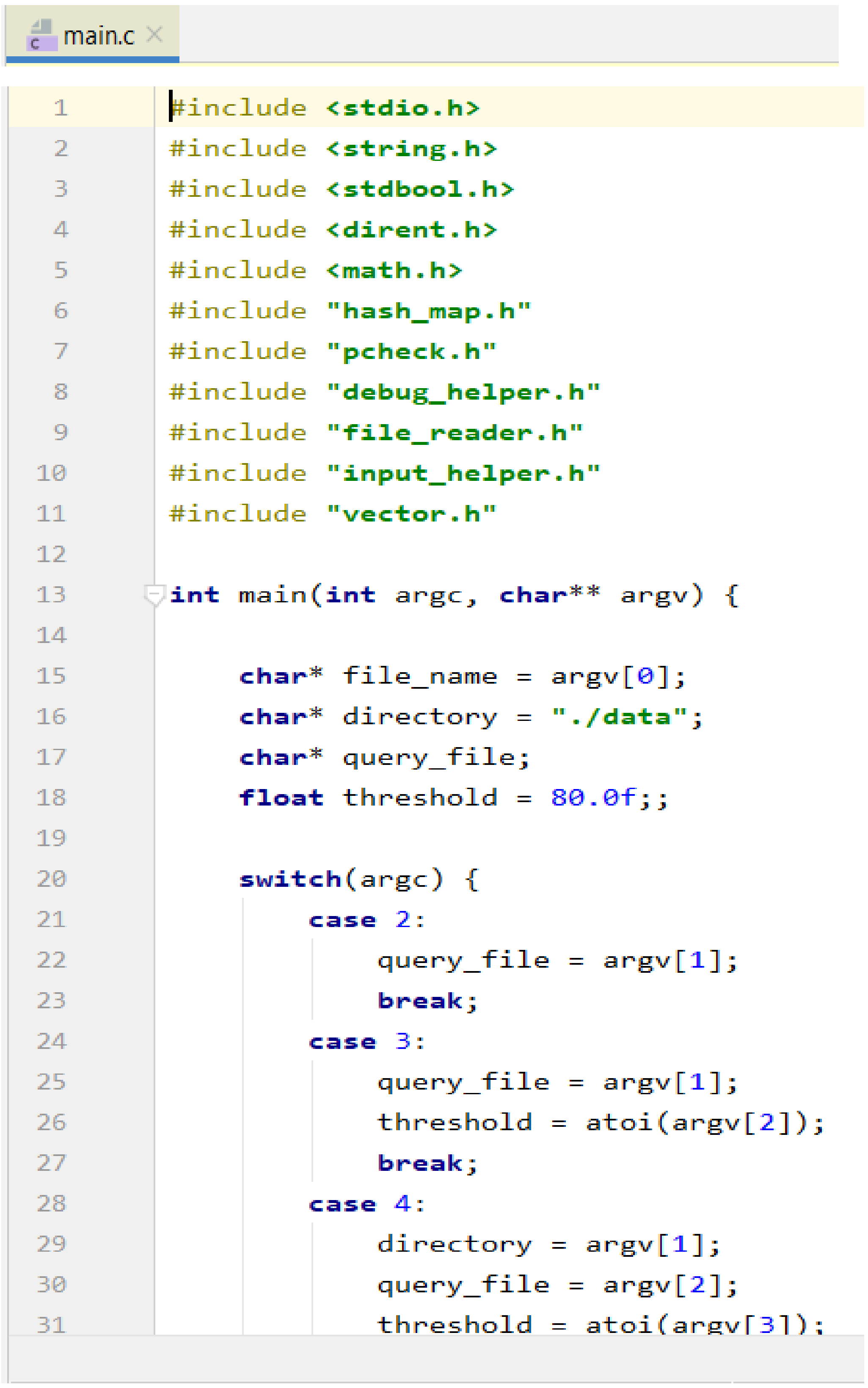
```
1. Begin
2. modQuery = 0
3. for (i = 0; i < length of query ; i++)
    3.1 modQuery = modQuery + (query -> data[i].value)$^2$
4. modQuery = $\sqrt{modQuery}$
5. modDocument = 0
6. for (i = 0; i < length of document ; i++)
    6.1 modDocument = modDocument + (document -> data[i].value)$^2$
7. modDocument = $\sqrt{modDocument}$
8. querytimesdocument = 0
9. for (i = 0; i < length of document ; i++)
    9.1 if (document)
10. End
```

compare_query_with_documents:

```
1. Begin
2. Declare Vecor bags, HashMap query_bag
3. Read all the document files and use bag_words to bag the words and store it in a HashMap and add it to the Vector bags
4. Bag the Query using bag_words and store it in query_bag
5. Normalize the bags using normalize_bags
6. Normalize the Query using normalize_bags
7. Compute TF*IDF using compute_tf_idf for bags
8. Compute TF*IDF using compute_tf_idf fro query
```

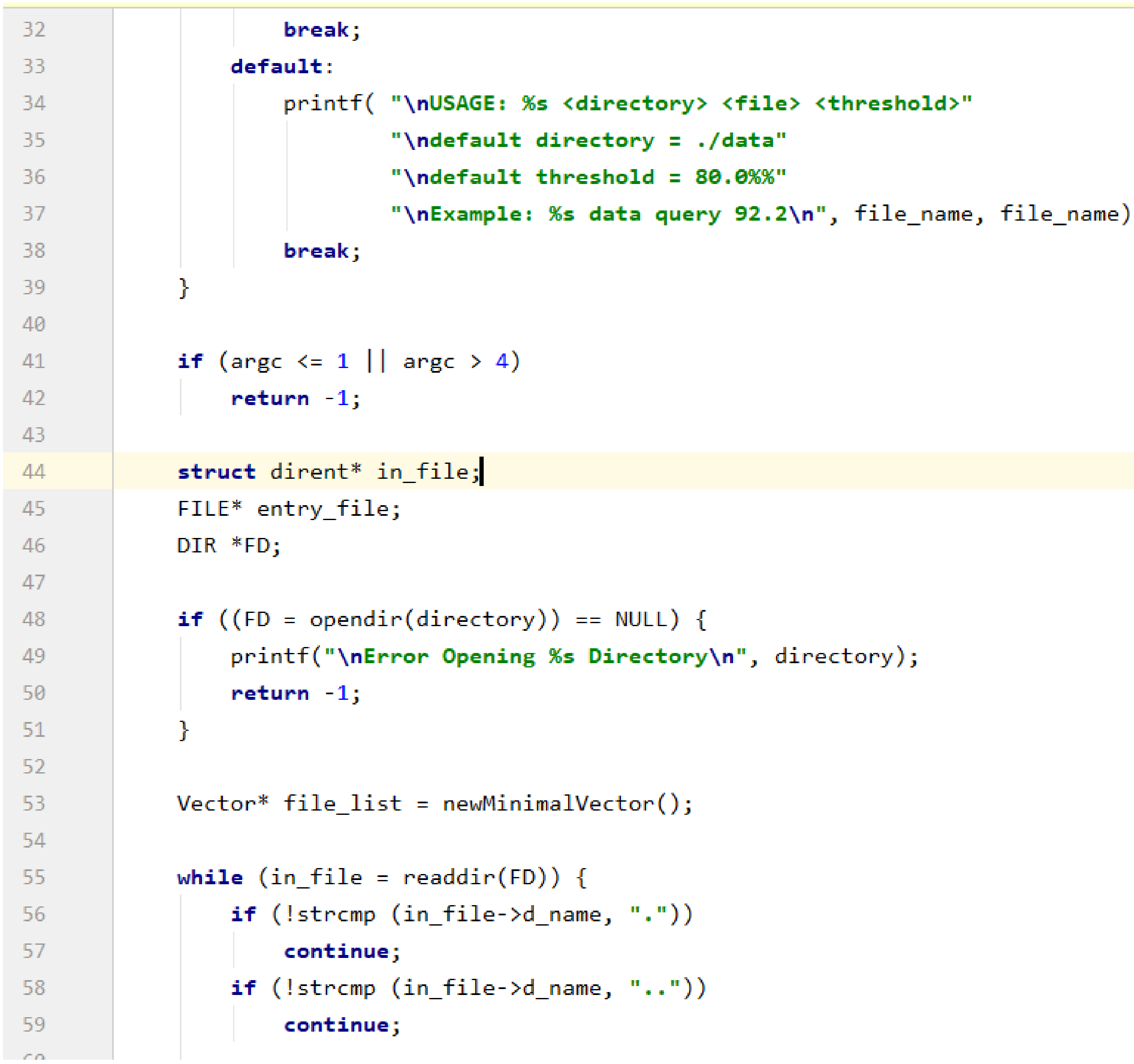
9. End

B 2.3 C Program:

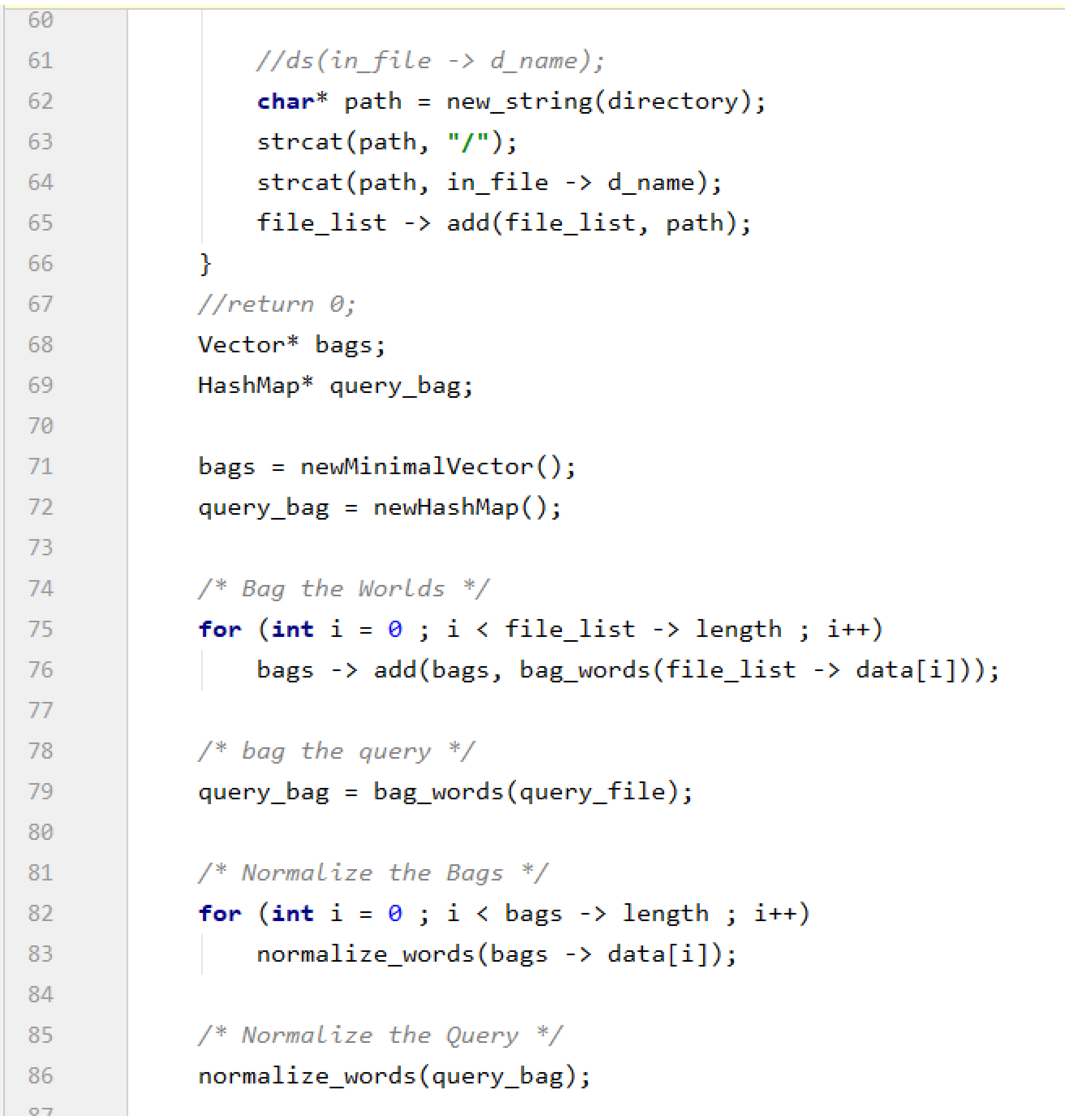


main.c

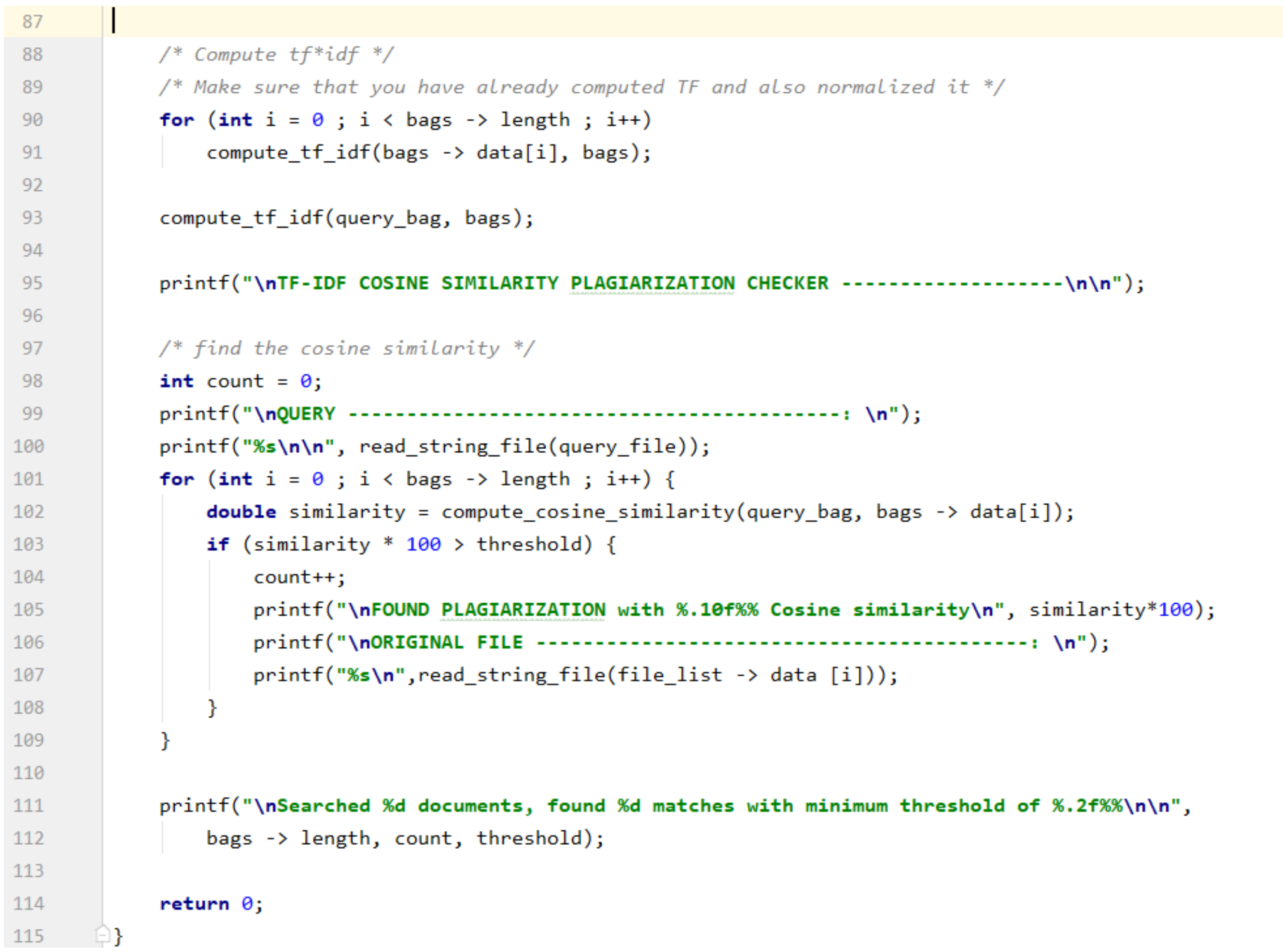
```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <dirent.h>
#include <math.h>
#include "hash_map.h"
#include "pcheck.h"
#include "debug_helper.h"
#include "file_reader.h"
#include "input_helper.h"
#include "vector.h"

int main(int argc, char** argv) {

    char* file_name = argv[0];
    char* directory = "./data";
    char* query_file;
    float threshold = 80.0f;;

    switch(argc) {
        case 2:
            query_file = argv[1];
            break;
        case 3:
            query_file = argv[1];
            threshold = atoi(argv[2]);
            break;
        case 4:
            directory = argv[1];
            query_file = argv[2];
            threshold = atoi(argv[3]);
```



```
            break;
        default:
            printf( "\nUSAGE: %s <directory> <file> <threshold>"
                    "\ndefault directory = ./data"
                    "\ndefault threshold = 80.0%%"
                    "\nExample: %s data query 92.2\n", file_name, file_name)
            break;
    }

    if (argc <= 1 || argc > 4)
        return -1;

    struct dirent* in_file;
    FILE* entry_file;
    DIR *FD;

    if ((FD = opendir(directory)) == NULL) {
        printf("\nError Opening %s Directory\n", directory);
        return -1;
    }

    Vector* file_list = newMinimalVector();

    while (in_file = readdir(FD)) {
        if (!strcmp (in_file->d_name, "."))
            continue;
        if (!strcmp (in_file->d_name, ".."))
            continue;
```

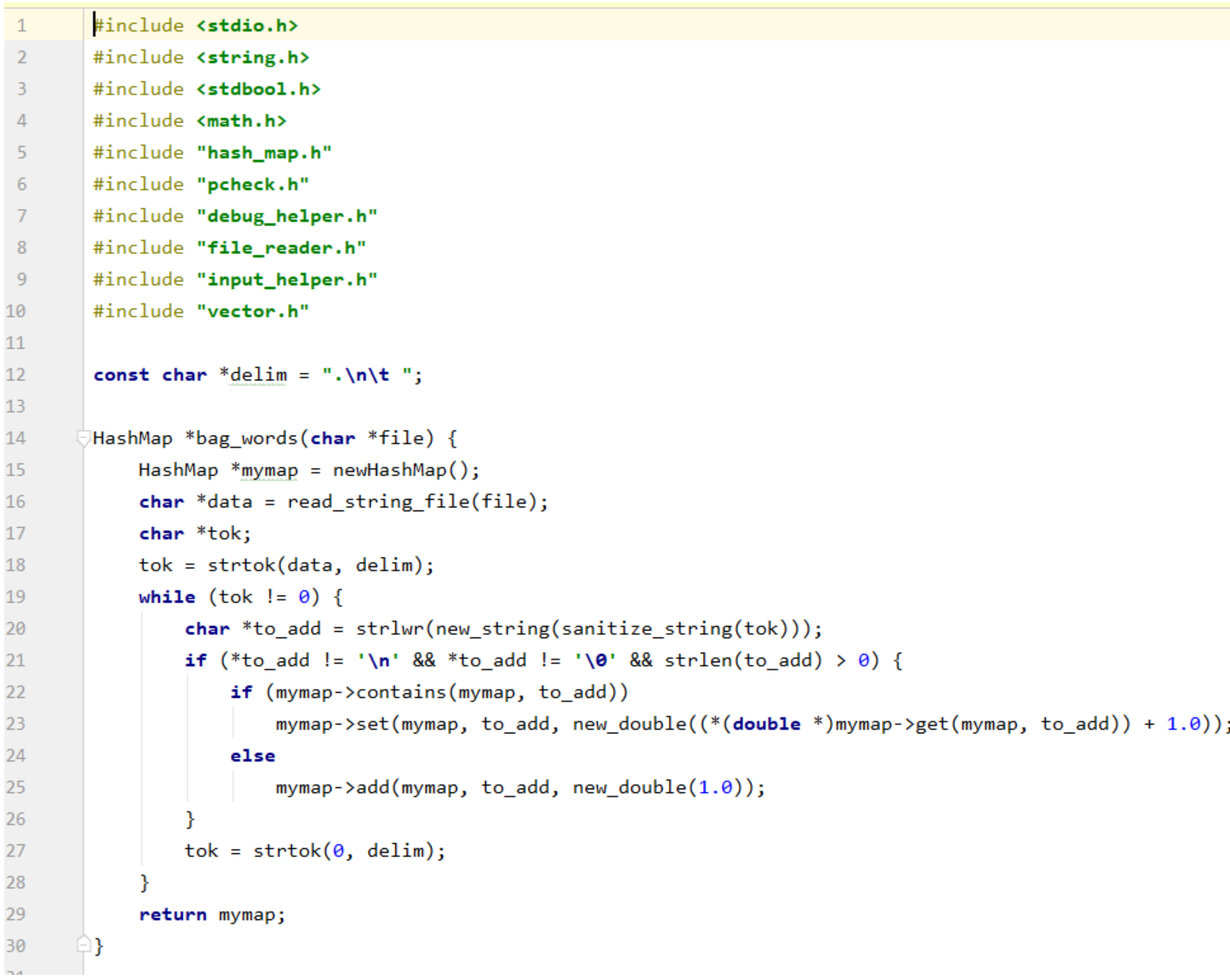


```c

        //ds(in_file -> d_name);
        char* path = new_string(directory);
        strcat(path, "/");
        strcat(path, in_file -> d_name);
        file_list -> add(file_list, path);
    }
    //return 0;
    Vector* bags;
    HashMap* query_bag;

    bags = newMinimalVector();
    query_bag = newHashMap();

    /* Bag the Worlds */
    for (int i = 0 ; i < file_list -> length ; i++)
        bags -> add(bags, bag_words(file_list -> data[i]));

    /* bag the query */
    query_bag = bag_words(query_file);

    /* Normalize the Bags */
    for (int i = 0 ; i < bags -> length ; i++)
        normalize_words(bags -> data[i]);

    /* Normalize the Query */
    normalize_words(query_bag);
```

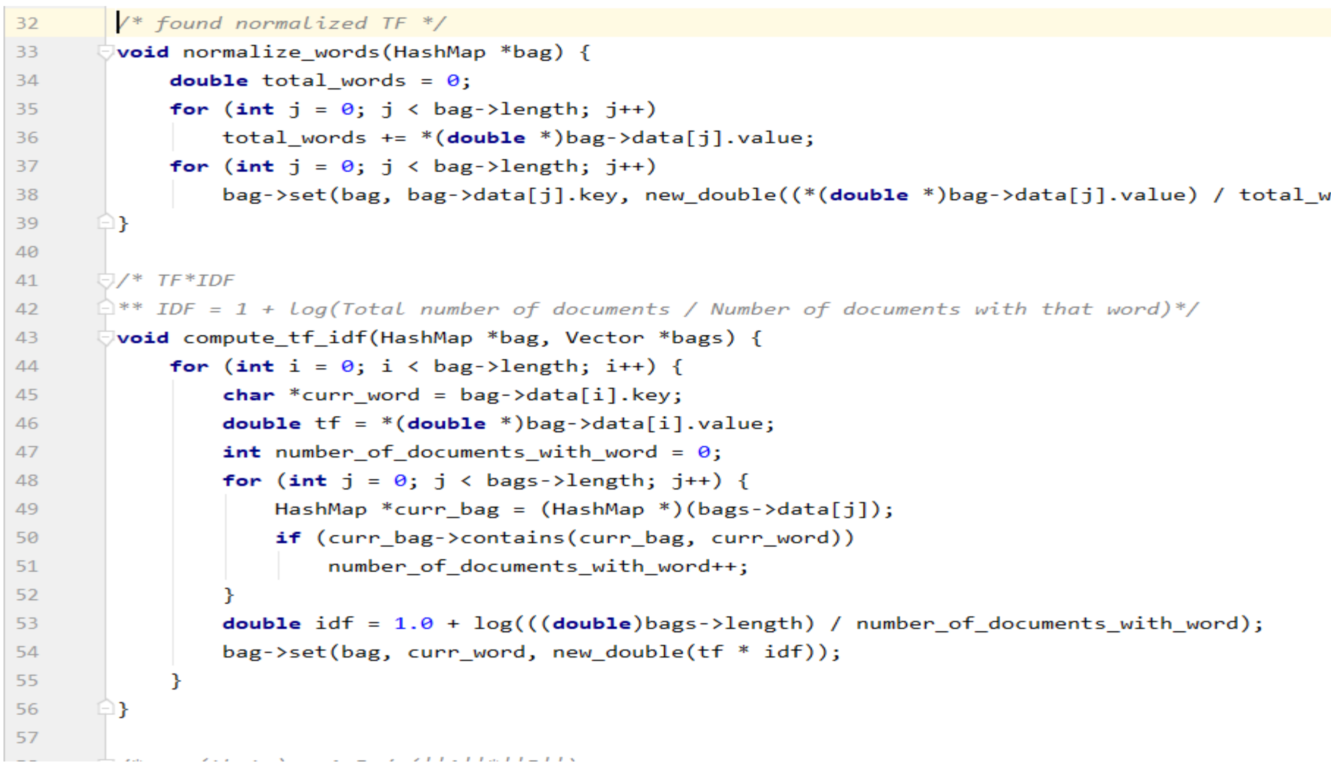


```c

    /* Compute tf*idf */
    /* Make sure that you have already computed TF and also normalized it */
    for (int i = 0 ; i < bags -> length ; i++)
        compute_tf_idf(bags -> data[i], bags);

    compute_tf_idf(query_bag, bags);

    printf("\nTF-IDF COSINE SIMILARITY PLAGIARIZATION CHECKER -------------------\n\n");

    /* find the cosine similarity */
    int count = 0;
    printf("\nQUERY ------------------------------------------: \n");
    printf("%s\n\n", read_string_file(query_file));
    for (int i = 0 ; i < bags -> length ; i++) {
        double similarity = compute_cosine_similarity(query_bag, bags -> data[i]);
        if (similarity * 100 > threshold) {
            count++;
            printf("\nFOUND PLAGIARIZATION with %.10f%% Cosine similarity\n", similarity*100);
            printf("\nORIGINAL FILE ------------------------------------------: \n");
            printf("%s\n",read_string_file(file_list -> data [i]));
        }
    }

    printf("\nSearched %d documents, found %d matches with minimum threshold of %.2f%%\n\n",
        bags -> length, count, threshold);

    return 0;
}
```

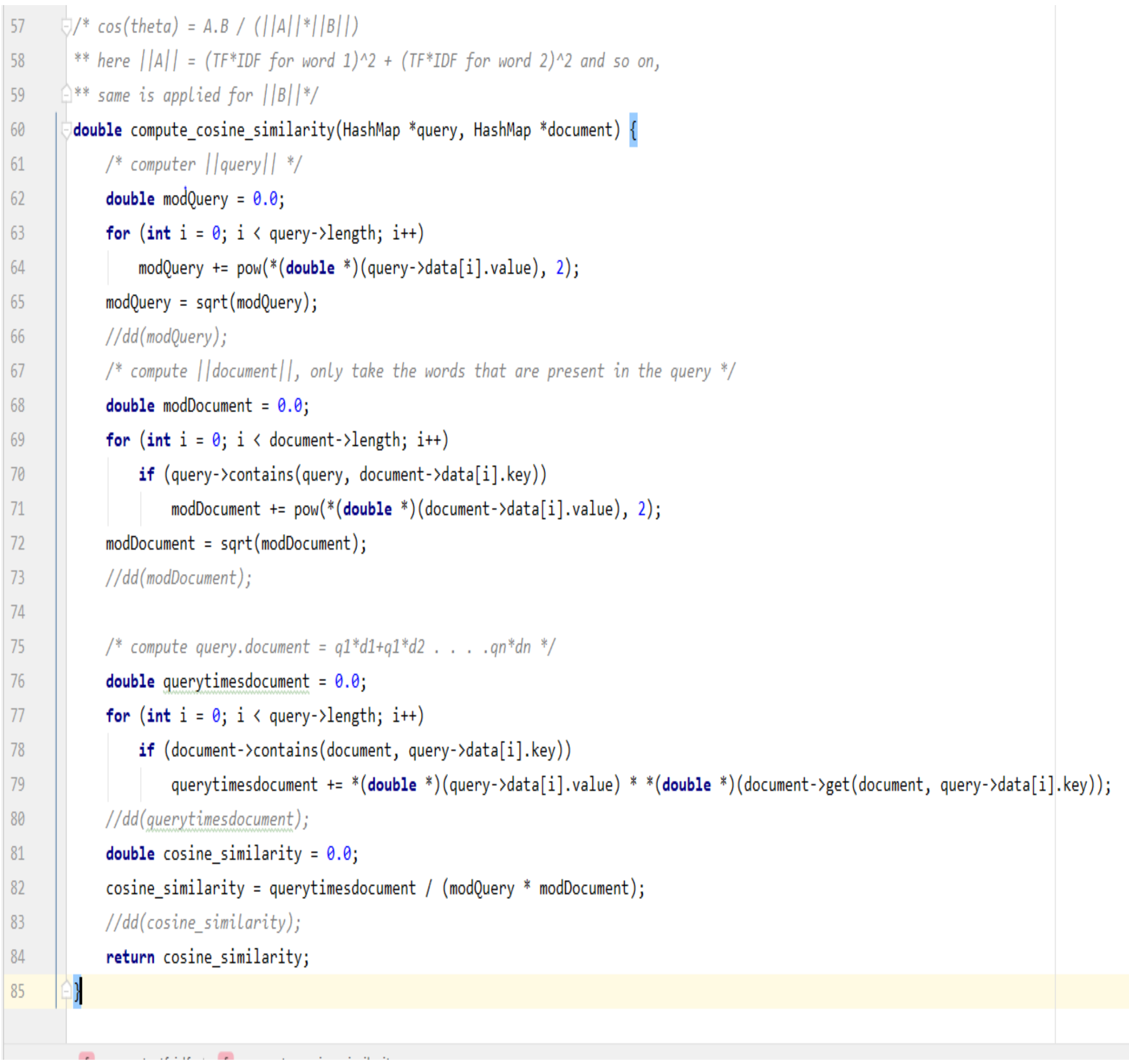
pcheck.c



```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include "hash_map.h"
#include "pcheck.h"
#include "debug_helper.h"
#include "file_reader.h"
#include "input_helper.h"
#include "vector.h"

const char *delim = ".\n\t ";

HashMap *bag_words(char *file) {
    HashMap *mymap = newHashMap();
    char *data = read_string_file(file);
    char *tok;
    tok = strtok(data, delim);
    while (tok != 0) {
        char *to_add = strlwr(new_string(sanitize_string(tok)));
        if (*to_add != '\n' && *to_add != '\0' && strlen(to_add) > 0) {
            if (mymap->contains(mymap, to_add))
                mymap->set(mymap, to_add, new_double((*(double *)mymap->get(mymap, to_add)) + 1.0));
            else
                mymap->add(mymap, to_add, new_double(1.0));
        }
        tok = strtok(0, delim);
    }
    return mymap;
}
```

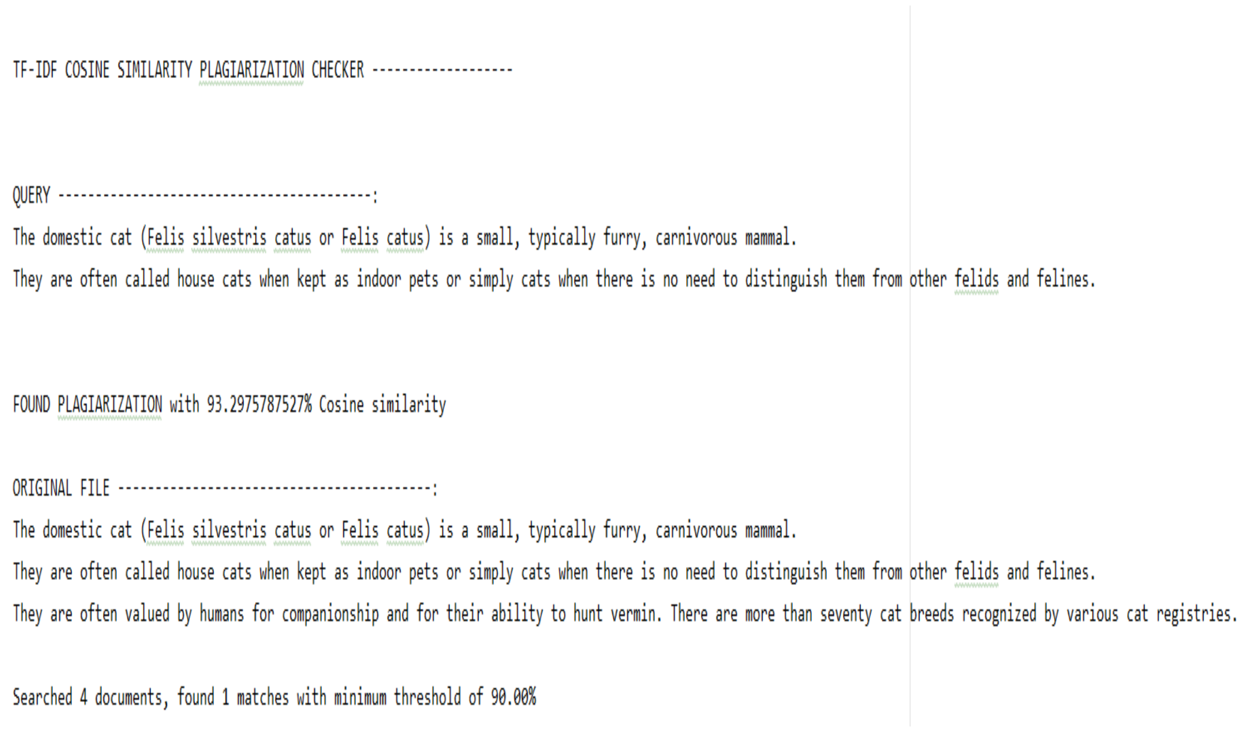


```
/* found normalized TF */
void normalize_words(HashMap *bag) {
    double total_words = 0;
    for (int j = 0; j < bag->length; j++)
        total_words += *(double *)bag->data[j].value;
    for (int j = 0; j < bag->length; j++)
        bag->set(bag, bag->data[j].key, new_double((*(double *)bag->data[j].value) / total_w
}

/* TF*IDF
** IDF = 1 + log(Total number of documents / Number of documents with that word)*/
void compute_tf_idf(HashMap *bag, Vector *bags) {
    for (int i = 0; i < bag->length; i++) {
        char *curr_word = bag->data[i].key;
        double tf = *(double *)bag->data[i].value;
        int number_of_documents_with_word = 0;
        for (int j = 0; j < bags->length; j++) {
            HashMap *curr_bag = (HashMap *)(bags->data[j]);
            if (curr_bag->contains(curr_bag, curr_word))
                number_of_documents_with_word++;
        }
        double idf = 1.0 + log(((double)bags->length) / number_of_documents_with_word);
        bag->set(bag, curr_word, new_double(tf * idf));
    }
}

```



```c
/* cos(theta) = A.B / (||A||*||B||)
** here ||A|| = (TF*IDF for word 1)^2 + (TF*IDF for word 2)^2 and so on,
** same is applied for ||B||*/
double compute_cosine_similarity(HashMap *query, HashMap *document) {
    /* computer ||query|| */
    double modQuery = 0.0;
    for (int i = 0; i < query->length; i++)
        modQuery += pow(*(double *)(query->data[i].value), 2);
    modQuery = sqrt(modQuery);
    //dd(modQuery);
    /* compute ||document||, only take the words that are present in the query */
    double modDocument = 0.0;
    for (int i = 0; i < document->length; i++)
        if (query->contains(query, document->data[i].key))
            modDocument += pow(*(double *)(document->data[i].value), 2);
    modDocument = sqrt(modDocument);
    //dd(modDocument);

    /* compute query.document = q1*d1+q1*d2 . . . .qn*dn */
    double querytimesdocument = 0.0;
    for (int i = 0; i < query->length; i++)
        if (document->contains(document, query->data[i].key))
            querytimesdocument += *(double *)(query->data[i].value) * *(double *)(document->get(document, query->data[i].key));
    //dd(querytimesdocument);
    double cosine_similarity = 0.0;
    cosine_similarity = querytimesdocument / (modQuery * modDocument);
    //dd(cosine_similarity);
    return cosine_similarity;
}
```

Output



```
TF-IDF COSINE SIMILARITY PLAGIARIZATION CHECKER ------------------



QUERY ----------------------------------------:
The domestic cat (Felis silvestris catus or Felis catus) is a small, typically furry, carnivorous mammal.
They are often called house cats when kept as indoor pets or simply cats when there is no need to distinguish them from other felids and felines.



FOUND PLAGIARIZATION with 93.2975787527% Cosine similarity


ORIGINAL FILE ----------------------------------------:
The domestic cat (Felis silvestris catus or Felis catus) is a small, typically furry, carnivorous mammal.
They are often called house cats when kept as indoor pets or simply cats when there is no need to distinguish them from other felids and felines.
They are often valued by humans for companionship and for their ability to hunt vermin. There are more than seventy cat breeds recognized by various cat registries.


Searched 4 documents, found 1 matches with minimum threshold of 90.00%
```

Explanation:

All the Source Documents are stored in a folder named data, which are text files, the query is also stored in a file named query that contains the potentially plagiarized document. These files are read, they are sanitized, i.e. the leading spaces, and trailing spaces are removes, they are split into words, bagged, and then each of the documents are converted to n-dimensional vector, the query is also converted to ndimensional vector, these vectors are then normalized, and the TF*IDF is calculated for each of them, the advantage of using TF*IDF is that common words across the documents are weighted down, i.e. they are assigned a lower score, and the unique words are weighted more, using logarithmic, now that we have a vector for each of the document and also the query, it can simply be compared using Dot Product, which is

$$A.B = ||A|| \times ||B|| \times \cos\theta$$

$$\cos\theta = \frac{A.B}{||A|| \times ||B||}$$

$\cos\theta$ will output a value from $[-1, 1]$, but for our implementation it won't output a value less than 0, hence the output of Cosine Similarity will be from $[0, 1]$, this can be mapped from $[0, 100]$ to get a percentage of potential plagiarisation.

It then checks which documents have a minimum plagiarisation which was passed as a command line argument to the program, and those plagiarized documents are displayed along with the query.

Bibliography

1. Francis Padillo (2016) A Data Structure to Speed-Up Machine Learning Algorithms on Massive Datasets.
2. Supreethi, K.P., Prasad, E.V.: Web Document Clustering Technique Using Case Grammar Structure. In: IEEE International Conference on Computational Intelligence and Multimedia Applications 2007 (2007).
3. US Patent US8375021B2, https://patents.google.com/patent/US8375021B2/en.
4. For further source codes regarding plagiarism question refer https://github.com/Deepakr120/plagarism-detector-using-c