Example Search Engine

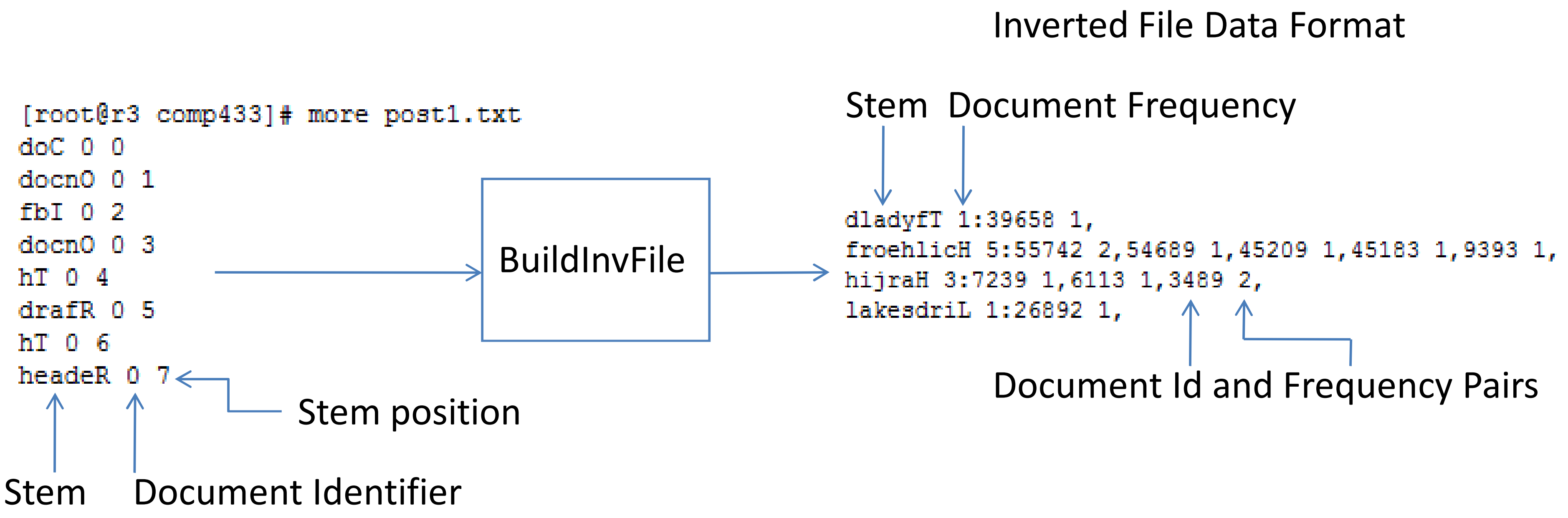
- directly extend it for your assignment;
- modify it for your assignment;
- use it as a reference to develop your own (possibly in another programming language like Java);
- don’t use it for your assignment.

Overview

- BuildInvFile.cpp
  - Convert data into inverted file format
- DocLen.cpp
  - Compute the document length
- Retrieval.cpp
  - Perform interactive retrieval
- Your Assignment

BuildInvFile.cpp

- Convert data in post1.txt into inverted file data format
  - Discard position information in post1.txt
  - Count the number of occurrences of each term (or stem)



```
[root@r3 comp433]# more post1.txt
doC 0 0
docnO 0 1
fbI 0 2
docnO 0 3
hT 0 4
drafR 0 5
hT 0 6
headeR 0 7
```

Stem position

Stem

Document Identifier

BuildInvFile

Inverted File Data Format

Stem

Document Frequency

```
dladyfT 1:39658 1,
froehlicH 5:55742 2,54689 1,45209 1,45183 1,9393 1,
hijraH 3:7239 1,6113 1,3489 2,
lakesdriL 1:26892 1,
```

Document Id and Frequency Pairs

BuildInvFile.cpp

```
[root@r3 comp433]# more post1.txt
doC 0 0
docn0 0 1
fbI 0 2
docn0 0 3
hT 0 4
drafR 0 5
hT 0 6
headeR 0 7
```

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include "IInvFile.h"

// Integrated Inverted Index (see lecture notes on Implementation)
IInvFile InvFile;

int main() {
        char tmp[10000];
        char str[1000];
        int docid;
        int loc;
        int cnt=0;
        FILE * fp = fopen("./post1.txt","rb");
        if (fp == NULL) {
                printf("Cannot open file \r\n");
                return 1;
                }

        // Initialize the Hash Table
        InvFile.MakeHashTable(13023973);

        while(fgets(tmp,10000,fp) != NULL) {
                // Get the stem, the document identifier and the location
                sscanf(tmp,"%s %d %d", &(str[0]), &docid, &loc);

                // Add posting into the Integrated Inverted index
                // See lecture notes on Implementation
                InvFile.Add(str, docid, 1);

                // Keep us informed about the progress
                cnt++;
                if ((cnt % 100000) == 0) printf("Added [%d]\r\n",cnt);
        }

        printf("Saving inverted file ...\r\n");
        InvFile.Save("InvFile.txt");

        InvFile.Clear();
        fclose(fp);
        return 0;
}
```

Include the integrated inverted index class

Declare an integrated inverted index object

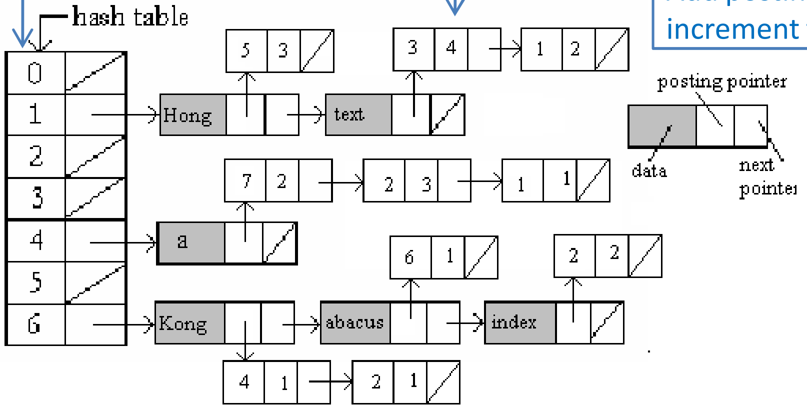
Store the stem or Index term

Open the file (if it doesn't exist, abort)

Create the hash table of the integrated inverted index object

Read one line of data

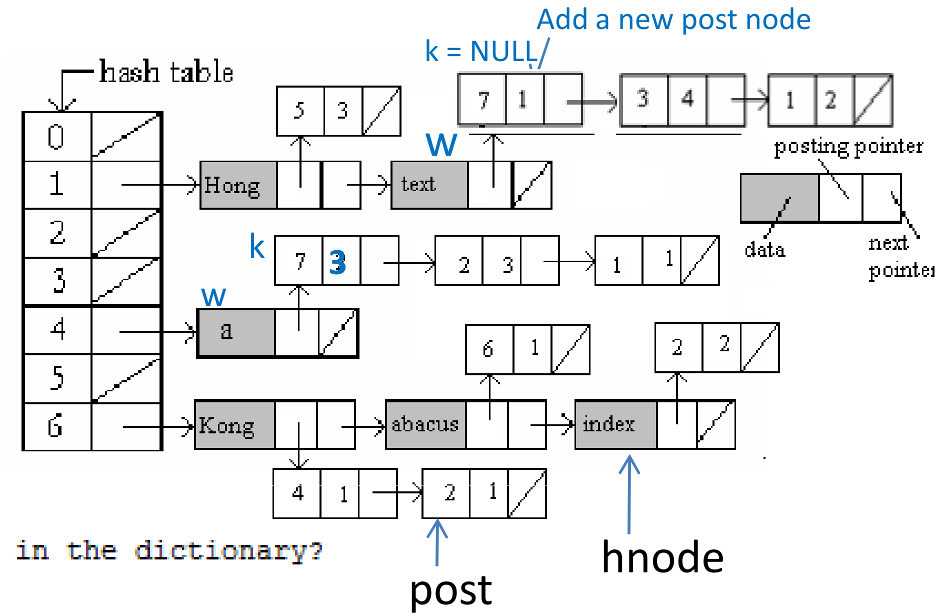
Add posting if not exist, else increment frequency count



Integrated Inverted Index Data Structure

Add Posting or Increment Frequency

- Add a hash node and possibly a post node if needed; else update frequency count
- Examples:
  - s = “text”, docid = 7, freq = 1
  - s = “a”, docid = 7, freq = 1

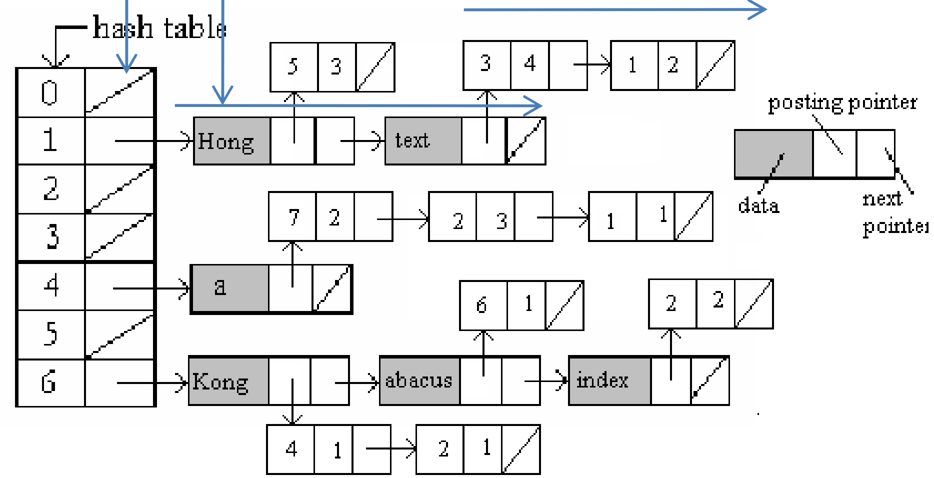


```
// Add a posting
post * IInvFile::Add(char * s, int docid, int freq) {
        hnode * w = Find(s);     // Does the stem exist in the dictionary?
        post * k = NULL;

        if (w == NULL) w = MakeHnode(s); // If not exist, create a new hash node
        else k = FindPost(w, docid);     // if exists, is the first posting the wanted one?

        if (k == NULL) {                        // no posting has the same docid
                k = w->posting;                 // push the data into the posting field
                w->posting = new post;          // create a new posting record
                w->posting->docid = docid;      // save the document id
                w->posting->freq = freq;        // save the term frequency
                w->df += 1;                     // keep track of the document frequency
                w->posting->next = k;           // push the data into the posting field
                }
        else    k->freq += freq;         // The posting exists, so add the freq to the freq field

        return k;
}
```

Save

```
// Save the data into the file
void IInvFile::Save(char * f) {
        FILE * fp = fopen(f, "wb");                   // open the file for writing
        hnode * w;
        post * k;
        if ((hsize > 0) && (htable != NULL)) {        // Is there a hash table?
                for(int i=0; i < hsize; i++) {        // For each hash table entry do
                        w = htable[i];                // Go through each hash node
                        while (w != NULL) {
                                fprintf(fp,"%s %d:", w->stem, w->df);
                                k = w->posting; // Go through each post node
                                while (k != NULL) {
                                        fprintf(fp,"%d %d,", k->docid, k->freq);
                                        k = k->next;     // next post node
                                        }
                                fprintf(fp,"\r\n");
                                w = w->next;             // next hash node
                        }
                }
        }
        fclose(fp);                                   // close the file
}
```

Declared in IInvFile class



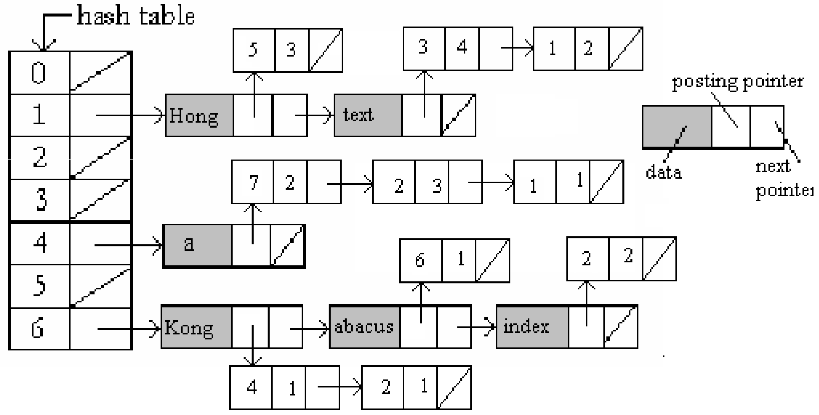
```
dladyfT 1:39658 1,
froehlicH 5:55742 2,54689 1,45209 1,45183 1,9393 1,
hijraH 3:7239 1,6113 1,3489 2,
lakesdriL 1:26892 1,
```

Overview

- BuildInvFile.cpp
  - Convert data into inverted file format
- DocLen.cpp
  - Compute the document length
- Retrieval.cpp
  - Perform interactive retrieval
- Your Assignment

Compute Document Length

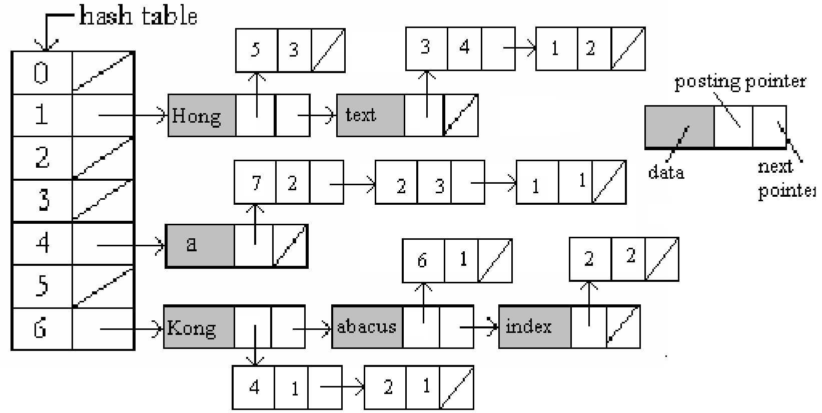
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include "IInvFile.h"

// Integrated Inverted Index (see lecture notes on Implementation)
IInvFile InvFile;

int main() {
        char tmp[10000];
        char str[1000];
        int docid;
        int loc;
        int cnt=0;

        // Initialize the Hash Table
        InvFile.MakeHashTable(13023973);

        printf("Loading Inverted File\r\n");
        InvFile.Load("InvFile.txt");
        printf("Creating Document Records (size = %d)\r\n", InvFile.MaxDocid+1);
        InvFile.MakeDocRec();   // allocate document records
        printf("Compute Document Lengths...\r\n");
        InvFile.DocLen(InvFile.Files);
        printf("Save Document Lengths\r\n");
        InvFile.SaveDocRec("InvFile.doc");

        return 0;
}
```



```
// Document related information
typedef struct _DocRec {
        float len;
        char * TRECID;
} DocRec;
```

Compute Document Length Function in IInvFile.cpp

- Similar to Save function
- File[] array stores the TRECID and the document length



```
void IInvFile::DocLen(DocRec File[]) {
        float idf;
        float idf2;
        hnode * w;
        post * k;

        if ((hsize > 0) && (htable != NULL)) {   // if there is a hash table (for open hashing)
                for(int i=0; i < hsize; i++) {   // Loop through every entry in the hash table
                        w = htable[i];           // Get the ith hash table entry
                        while (w != NULL) {      // Loop through each hash node (hnode) in the linked list
                                idf = GetIDF(w->df);    // Get the IDF value based on the document frequency, df
                                idf2 = idf * idf;       // square of IDF
                                k = w->posting;         // Loop through each posting in the posting list
                                while (k != NULL) {
                                        if (k->docid <= MaxDocid)
                                                File[k->docid].len += idf2 * (float) (k->freq * k->freq); // TF*IDF square
                                        else
                                                printf("DocLen Error: Docid = %d > Max = %d\r\n",k->docid, MaxDocid);
                                        k = k->next;    // next posting
                                        }
                                w = w->next;            // next hash node in the linked list
                        }
                }
        }
        else printf("Doclen aborted: no hash table\r\n");
        for(int i=0; i <=MaxDocid; i++) File[i].len = (float) sqrt((double) File[i].len);
}
```

Loop through hash table

Loop through hash node linked list

Loop through post node linked list

Declared in IInvFile class

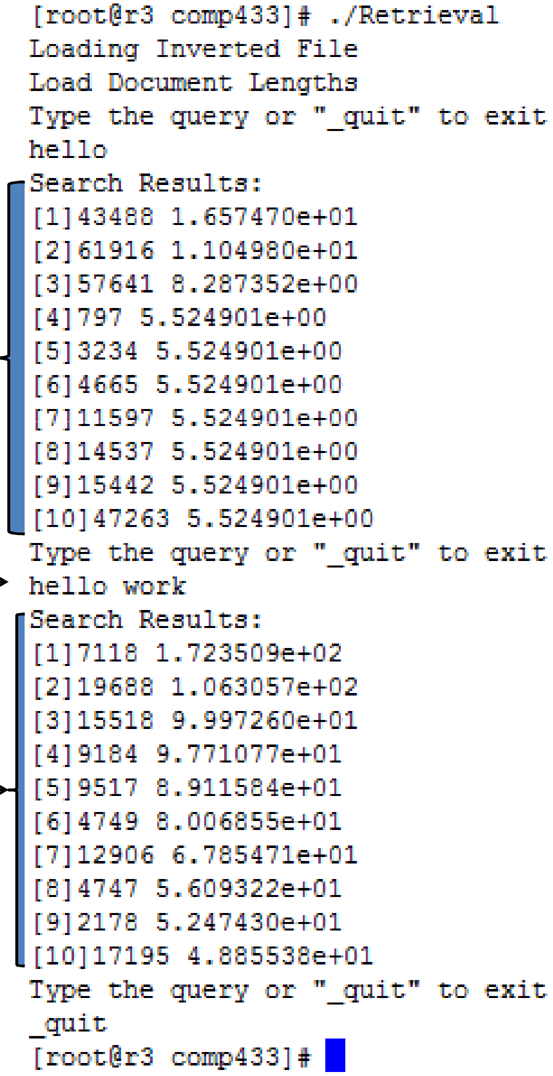
$$|D|_2 = \sqrt{\sum_{t \in V} (IDF(t) \times F(t,D))^2}$$

Overview

- BuildInvFile.cpp
  - Convert data into inverted file format
- DocLen.cpp
  - Compute the document length
- Retrieval.cpp
  - Perform interactive retrieval
- Your Assignment

Retrieval: Example Run

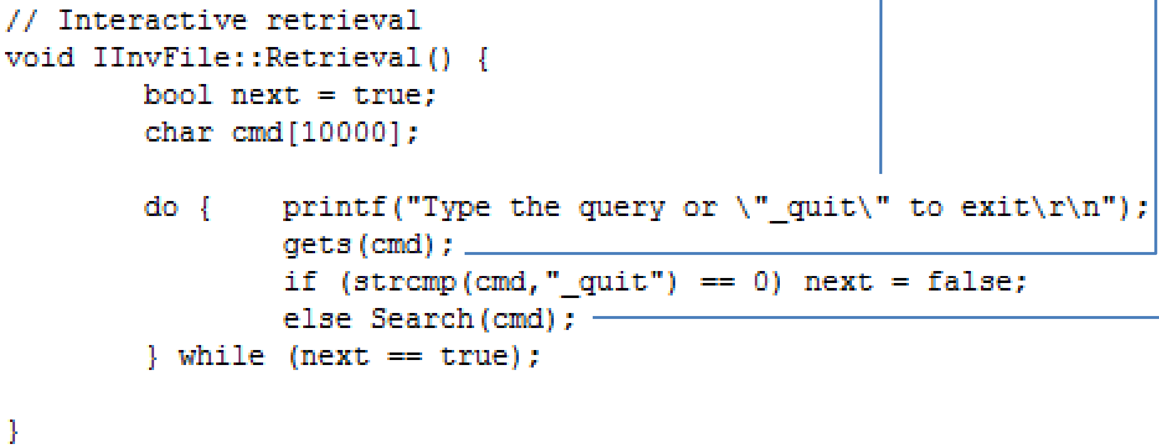
- Make sure you have created the InvFile.txt (using BuildInvFile) and InvFile.doc (using DocLen)
- Type ./Retrieval to run the program
- Display:
  - Loading …
  - Type the query …
- Type the query “hello”
- Top 10 results
- Type the query “hello work”
- Top 10 results
- Type “_quit” to exit



```
[root@r3 comp433]# ./Retrieval
Loading Inverted File
Load Document Lengths
Type the query or "_quit" to exit
hello
Search Results:
[1]43488 1.657470e+01
[2]61916 1.104980e+01
[3]57641 8.287352e+00
[4]797 5.524901e+00
[5]3234 5.524901e+00
[6]4665 5.524901e+00
[7]11597 5.524901e+00
[8]14537 5.524901e+00
[9]15442 5.524901e+00
[10]47263 5.524901e+00
Type the query or "_quit" to exit
hello work
Search Results:
[1]7118 1.723509e+02
[2]19688 1.063057e+02
[3]15518 9.997260e+01
[4]9184 9.771077e+01
[5]9517 8.911584e+01
[6]4749 8.006855e+01
[7]12906 6.785471e+01
[8]4747 5.609322e+01
[9]2178 5.247430e+01
[10]17195 4.885538e+01
Type the query or "_quit" to exit
_quit
[root@r3 comp433]#
```

Retrieval.cpp

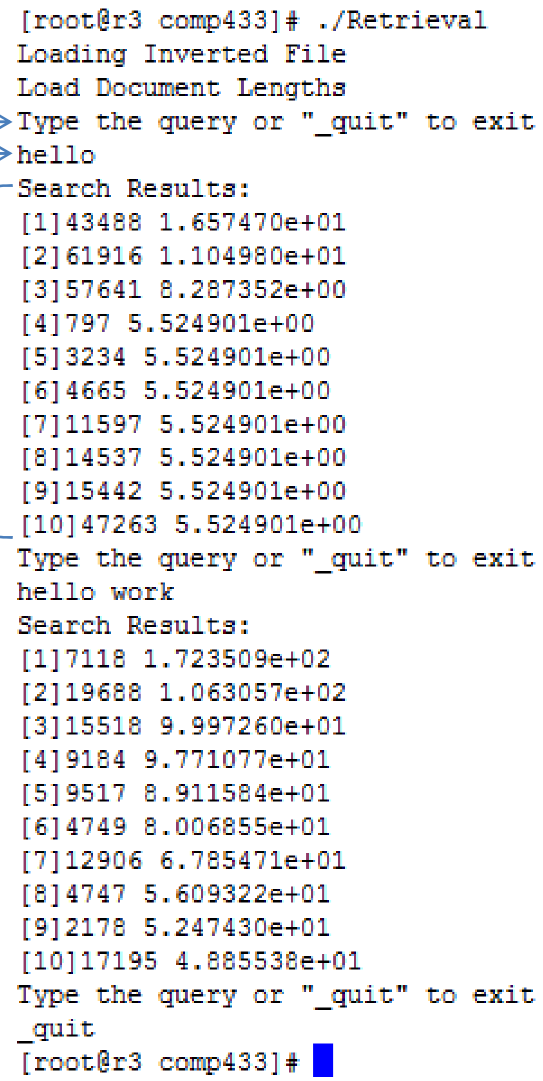
- Gets a line that contains the query
- Carry out the search



```
// Interactive retrieval
void IInvFile::Retrieval() {
        bool next = true;
        char cmd[10000];

        do {    printf("Type the query or \"_quit\" to exit\r\n");
                gets(cmd);
                if (strcmp(cmd,"_quit") == 0) next = false;
                else Search(cmd);
        } while (next == true);

}
```

- The search results are stored in an array of retrieval records:

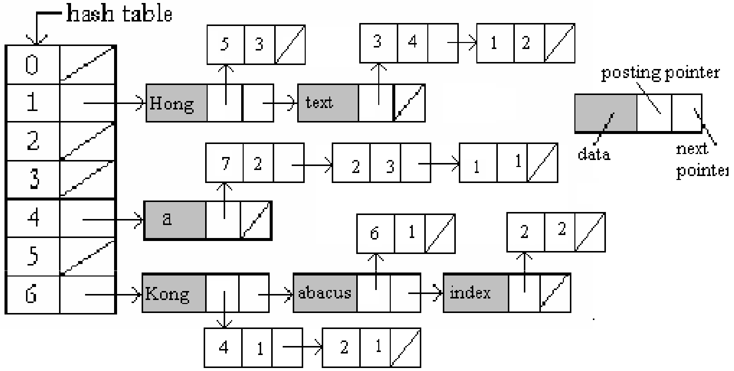
```
typedef struct _RetRec {
        int docid;          // document identifier
        float sim;          // similarity score
} RetRec;
```



```
[root@r3 comp433]# ./Retrieval
Loading Inverted File
Load Document Lengths
Type the query or "_quit" to exit
hello
Search Results:
[1]43488 1.657470e+01
[2]61916 1.104980e+01
[3]57641 8.287352e+00
[4]797 5.524901e+00
[5]3234 5.524901e+00
[6]4665 5.524901e+00
[7]11597 5.524901e+00
[8]14537 5.524901e+00
[9]15442 5.524901e+00
[10]47263 5.524901e+00
Type the query or "_quit" to exit
hello work
Search Results:
[1]7118 1.723509e+02
[2]19688 1.063057e+02
[3]15518 9.997260e+01
[4]9184 9.771077e+01
[5]9517 8.911584e+01
[6]4749 8.006855e+01
[7]12906 6.785471e+01
[8]4747 5.609322e+01
[9]2178 5.247430e+01
[10]17195 4.885538e+01
Type the query or "_quit" to exit
_quit
[root@r3 comp433]# 
```

C++ Program to handle One Query

- The retrieval records are stored in “result” array (the result set).
- Process one query term at a time: get one term, get its postings, combine them into the result set

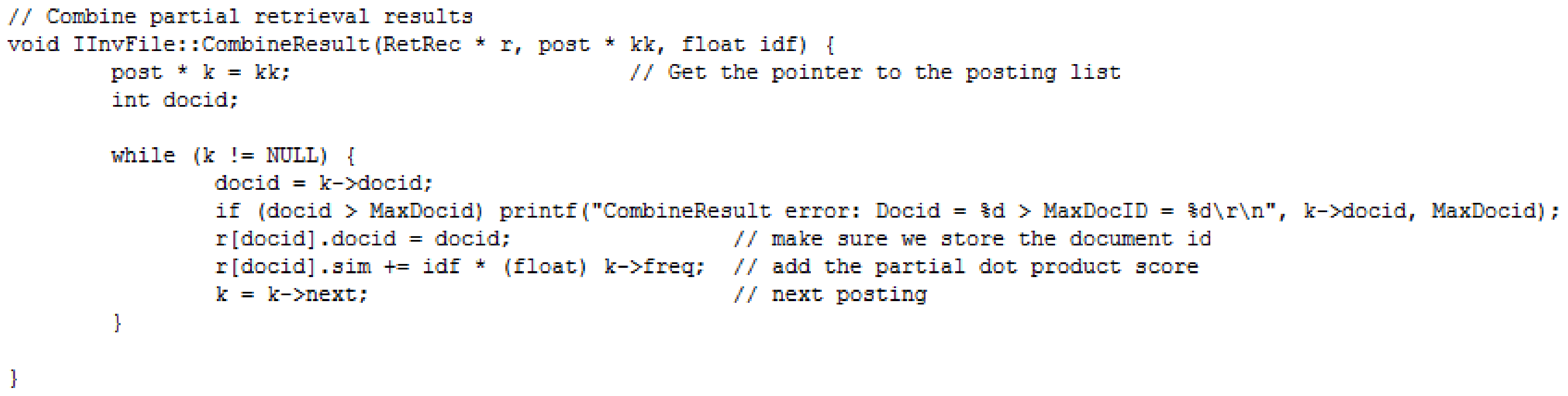


```
// Perform retrieval
void IInvFile::Search(char * q) {
        char * s = q;
        char * w;
        bool next = true;
        hnode * h;
        // Initialize the result set
        if (result != NULL) free(result);
        result = (RetRec *) calloc(MaxDocid+1, sizeof(RetRec));

        do {    w = s;                                  // Do searching
                s = GotoNextWord(s);                    // Delimit the term
                if (s == NULL) next = false;            // If no more terms, exit
                else { if (*s != '\0') *(s-1) = '\0';   // If not the last term, delimit the term
                        Stemming.Stem(w);               // Stem the term w
                        h = Find(w);                    // Find it in the integrated inverted index
                        if (h != NULL)                  // Add the scores to the result set
                                CombineResult(result, h->posting, GetIDF(h->df));
                        else if (strlen(w) > 0) printf("Query term does not exist <%s>\r\n",w);
                }
        } while (next == true);                         // More query terms to handle?

        PrintTop(result, 10);                           // Print top 10 retrieved results
}
```

Declared in IInvFile class

C++ Function that Combines Partial Results

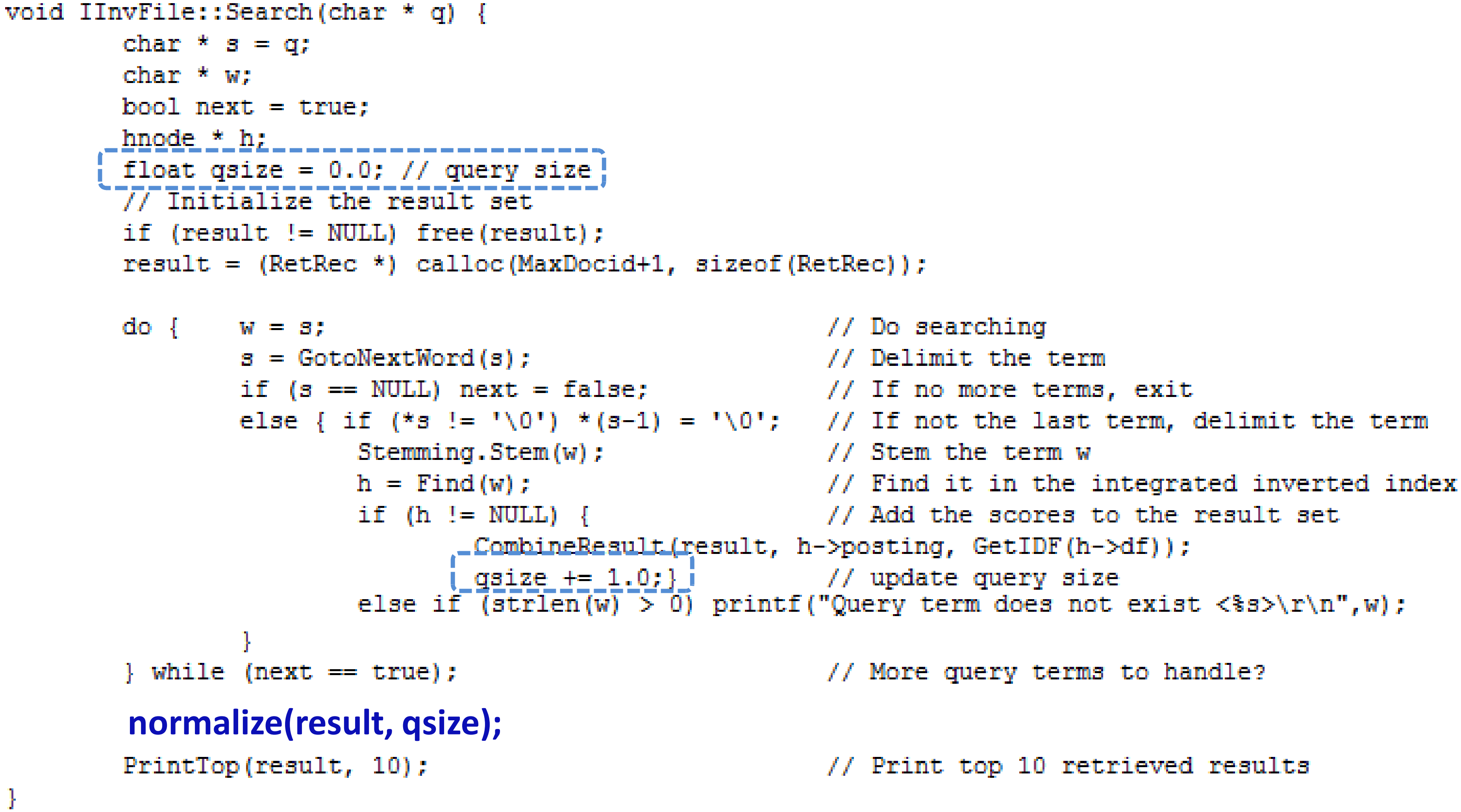
- Combines the retrieval results with the existing retrieval results that are stored in the result set (`result`)
- The partial dot product score is computed by adding it with the TF*IDF value
  - The IDF value of the term is given (idf): see previous slide
  - The term frequency is obtained by “k->freq”
- The “docid” variable is used for convenience



```
// Combine partial retrieval results
void IInvFile::CombineResult(RetRec * r, post * kk, float idf) {
        post * k = kk;                          // Get the pointer to the posting list
        int docid;

        while (k != NULL) {
                docid = k->docid;
                if (docid > MaxDocid) printf("CombineResult error: Docid = %d > MaxDocID = %d\r\n", k->docid, MaxDocid);
                r[docid].docid = docid;                 // make sure we store the document id
                r[docid].sim += idf * (float) k->freq;  // add the partial dot product score
                k = k->next;                            // next posting
        }

}
```

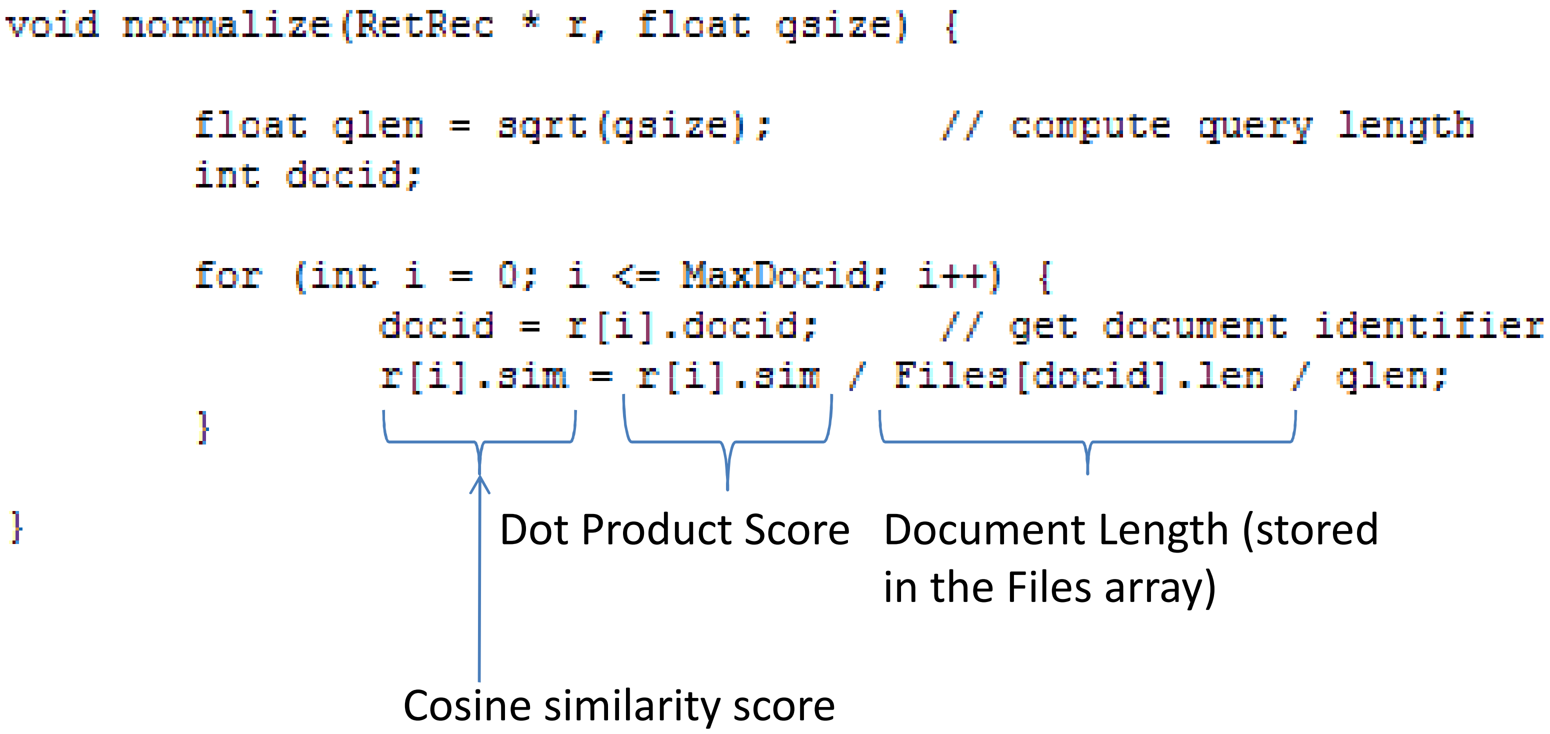
Compute the Cosine Similarity Score

- Modify the retrieval program by adding a function called “normalize”



```
void IInvFile::Search(char * q) {
        char * s = q;
        char * w;
        bool next = true;
        hnode * h;
        float qsize = 0.0; // query size
        // Initialize the result set
        if (result != NULL) free(result);
        result = (RetRec *) calloc(MaxDocid+1, sizeof(RetRec));

        do {    w = s;                                      // Do searching
                s = GotoNextWord(s);                        // Delimit the term
                if (s == NULL) next = false;                // If no more terms, exit
                else { if (*s != '\0') *(s-1) = '\0';       // If not the last term, delimit the term
                        Stemming.Stem(w);                   // Stem the term w
                        h = Find(w);                        // Find it in the integrated inverted index
                        if (h != NULL) {                    // Add the scores to the result set
                                CombineResult(result, h->posting, GetIDF(h->df));
                                qsize += 1.0;}              // update query size
                        else if (strlen(w) > 0) printf("Query term does not exist <%s>\r\n",w);
                }
        } while (next == true);                             // More query terms to handle?
        normalize(result, qsize);
        PrintTop(result, 10);                               // Print top 10 retrieved results
}
```

normalize(.)



```
void normalize(RetRec * r, float qsize) {

        float qlen = sqrt(qsize);        // compute query length
        int docid;

        for (int i = 0; i <= MaxDocid; i++) {
                docid = r[i].docid;      // get document identifier
                r[i].sim = r[i].sim / Files[docid].len / qlen;
        }

}
```

Dot Product Score

Document Length (stored in the Files array)

Cosine similarity score

For Your Assignment

- Read the query file
  - A query occupies one line and has a query ID

TREC Queries

```
601 Turkey Iraq water
602 Czech, Slovak sovereignty
603 Tobacco cigarette lawsuit
```

- You need to print the TRECID
  - You need to read the association between TRECID and the document identifier from the “file.txt” (in file1.ZIP)

```
0 156 @ FBIS3-1001 f:\assign\data\FBIS3-1001
1 270 @ FBIS3-10010 f:\assign\data\FBIS3-10010
2 156 @ FBIS3-10012 f:\assign\data\FBIS3-10012
3 267 @ FBIS3-10026 f:\assign\data\FBIS3-10026
4 260 @ FBIS3-10030 f:\assign\data\FBIS3-10030
5 932 @ FBIS3-10033 f:\assign\data\FBIS3-10033
```

Document identifier   TRECID

  - Store the TRECID string for each document record in Files[.] array
  - Output the search result in the TREC output format (including the TRECID rather than the document identifier) for the evaluation program to run
- Print the top 1000 ranked documents

```
Search Results:
[1]43488 1.657470e+01
[2]61916 1.104980e+01
[3]57641 8.287352e+00
[4]797 5.524901e+00
[5]3234 5.524901e+00
[6]4665 5.524901e+00
[7]11597 5.524901e+00
[8]14537 5.524901e+00
[9]15442 5.524901e+00
[10]47263 5.524901e+00
```

Change these document identifiers into TRECIDs for your assignment (Change the format too)

```
301 Q0 FBIS-325 1 27.8 HKPU-1
301 Q0 FBIS-178 2 23.1 HKPU-1
…
```

TREC Format: Query ID TRECID Rank Similarity Score RunID

For Your Assignment

- Read the TRECID from file.txt
  - Call: InvFile.ReadTRECID(“file.txt”);
- Program (Add this in IInvFile.cpp):
  - Read one line at a time and add TRECID each time

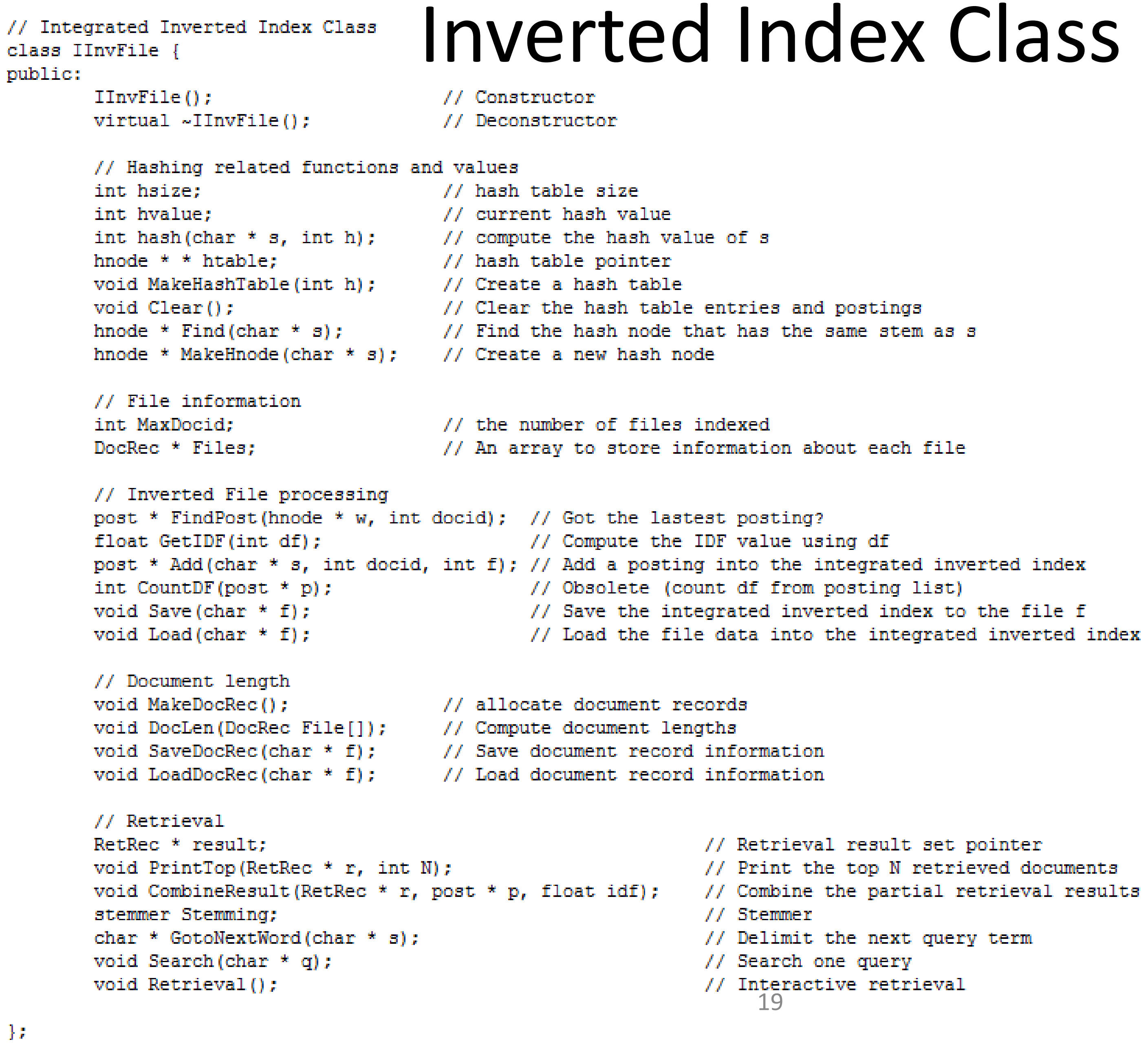
```
void IInvFile::ReadTRECID(char * f) {
    char line[10000]; char str[1000]; char TRECID[1000];
    FILE * fp = fopen(f, “rb”);
    if (fp == NULL) {printf(“Error : no file <%s>”,f); return;}
    while (fgets(line,10000,fp) != NULL) {
        sscanf(line,”%d %d %s %s”,&docid, &len, &(str[0]), &(TRECID[0]));
        Files[docid].TRECID = strdup(TRECID);
        }
    fclose(fp);
}
```

Declared in IInvFile class

docid   len   str   TRECID

```
0 156 @ FBIS3-1001 f:\assign\data\FBIS3-1001
1 270 @ FBIS3-10010 f:\assign\data\FBIS3-10010
2 156 @ FBIS3-10012 f:\assign\data\FBIS3-10012
3 267 @ FBIS3-10026 f:\assign\data\FBIS3-10026
4 260 @ FBIS3-10030 f:\assign\data\FBIS3-10030
5 932 @ FBIS3-10033 f:\assign\data\FBIS3-10033
```

C++ Class Declaration of the Integrated Inverted Index Class



```
// Integrated Inverted Index Class
class IInvFile {
public:
        IInvFile();                     // Constructor
        virtual ~IInvFile();            // Deconstructor

        // Hashing related functions and values
        int hsize;                      // hash table size
        int hvalue;                     // current hash value
        int hash(char * s, int h);      // compute the hash value of s
        hnode * * htable;               // hash table pointer
        void MakeHashTable(int h);      // Create a hash table
        void Clear();                   // Clear the hash table entries and postings
        hnode * Find(char * s);         // Find the hash node that has the same stem as s
        hnode * MakeHnode(char * s);    // Create a new hash node

        // File information
        int MaxDocid;                   // the number of files indexed
        DocRec * Files;                 // An array to store information about each file

        // Inverted File processing
        post * FindPost(hnode * w, int docid);  // Got the lastest posting?
        float GetIDF(int df);                   // Compute the IDF value using df
        post * Add(char * s, int docid, int f); // Add a posting into the integrated inverted index
        int CountDF(post * p);                  // Obsolete (count df from posting list)
        void Save(char * f);                    // Save the integrated inverted index to the file f
        void Load(char * f);                    // Load the file data into the integrated inverted index

        // Document length
        void MakeDocRec();              // allocate document records
        void DocLen(DocRec File[]);     // Compute document lengths
        void SaveDocRec(char * f);      // Save document record information
        void LoadDocRec(char * f);      // Load document record information

        // Retrieval
        RetRec * result;                                        // Retrieval result set pointer
        void PrintTop(RetRec * r, int N);                       // Print the top N retrieved documents
        void CombineResult(RetRec * r, post * p, float idf);    // Combine the partial retrieval results
        stemmer Stemming;                                       // Stemmer
        char * GotoNextWord(char * s);                          // Delimit the next query term
        void Search(char * q);                                  // Search one query
        void Retrieval();                                       // Interactive retrieval

};
```