CSI 2290 Final Project:

Information Retrieval System

Contributors:

Aaron Kenward

Cameron Vogeli

Eric Daulo

Weiye Shi

Carlos Buenaventura

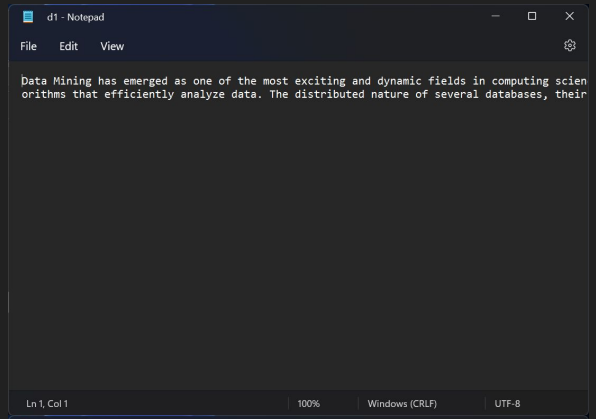
John Musa

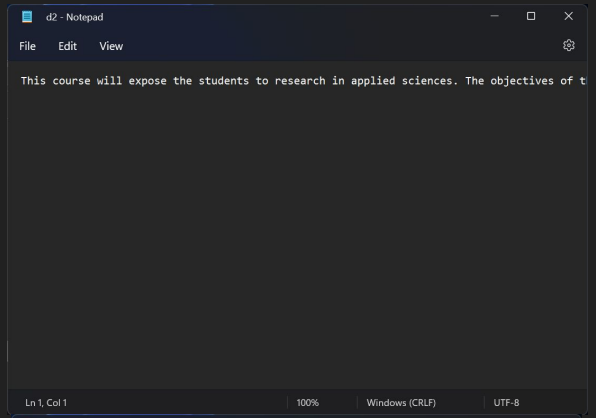
Introduction

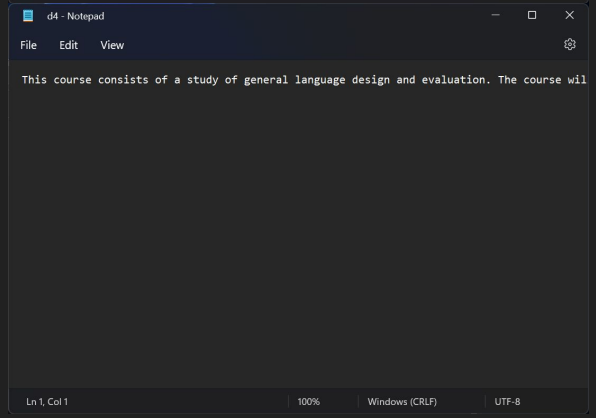
Steps of project:

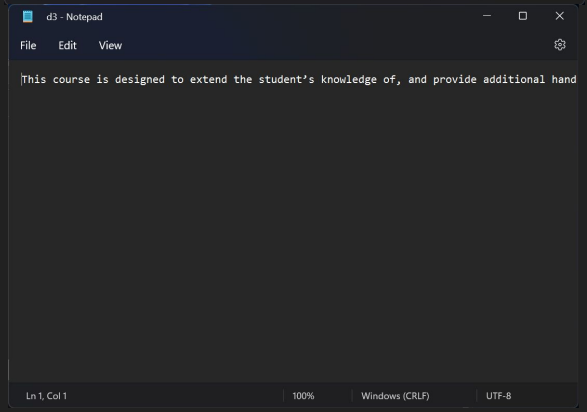
1. Putting text into .txt files.
2. Creating lists of token/words
3. Eliminate stop words (from “stopwords.txt” file)
4. Eliminate special characters (from “specialcharacters.txt” file)
5. Sort words alphabetically
6. Store words in a new file
7. Compute frequency and weight

Files with text









Listing all the Data from the Input Files (Main Function)

```
FILE *d1, *d2, *d3, *d4;

d1 = fopen("../assets/inputs/d1.txt", "r");

.....

d4 = fopen("../assets/inputs/d4.txt", "r");
```

The natural first step in reading from the input files was to open them using fopen and filestream pointers. We then would create a conditional statement in case the input file is null (doesn't exist) to return an error message, and to run init function on the input if it can read from the file. Our code for starting the process of alphabetizing, counting, removing (special characters and stop words), and tokenizing was as follows:

```
init( d1, "../assets/outputs/Tokenizedd1.txt" );

…

init( d4, "../assets/outputs/Tokenizedd4.txt" );
```

Structures

```
typedef struct Str
{
        double numTimes;
        char word[50];
}Str;
```

Str structure

```
typedef struct
{
        Str* data;
        int listsize;
        int length;
}Sqlist;
```

Sqlist structure

For the sake of the tokenization process, two structures were created: one for storing words and their frequencies, and one for storing the list of strings used in the early part of our program.

Each word is initially stored in a 50-character array within each “str” structure, along with a double value which denotes its # of appearances.

The other defined structure contains a “str-type” (the previously defined structure) pointer for the words, an integer value for the size of the list, and another integer for length.

These two structure definitions form the backbone of the execution of our code.

initSqList: First component of reading from file

Function Definition for initSqList:

```
int initSqlist(Sqlist* L, int maxsize) {
        L->data = (Str*)malloc(maxsize * sizeof(Str));
        if (L->data == NULL)
        {
                printf("Space application failed!"); exit(0);
        }
        L->listsize = maxsize;
        L->length = 0;
        return 1;
}
```

Explanation:

This is the definition for a function called initSqList, which has a structure of type Sqlist (previous slide) named L as its first argument, and a constant integer named maxsize as the second argument. This function is necessary for initializing the values for the list we will use. This code starts by dynamically allocating data that is populated with the data field from L. If the data can't be read (NULL), then an error message displays and the function exits. Otherwise, the listsize field of L is set to the maxsize integer from the function argument, and the length field of L is set to 0.

Printing Words into the List

After creating the list with the previous function, we then have to read the data from the input files into the list. To do this, we have a function called **getWords**. **getWords** starts by creating character arrays (strings): one for storing the text from a file [called string], one as a buffer for storing the string before stop words and special characters are removed, and one for storing the stop words. The function runs through a loop using fgets until a null condition is reached (end of file).

In the loop, it first uses strcpy() to copy the contents of the buffer into string. This string is then delimited into words by using strtok() and setting spaces as a delimiter. A character-type pointer is then set equal to this delimited string. After this, a while loop is used to read through every line of the stopwords.txt file and it uses strcmp to see if any entries in the list are stop words. If they are not, then they get committed to L to be parsed and tokenized, which we will discuss later.

Parsing and Removing Special Characters

This function works by copying the data previously obtained through getWords into a buffer array, then (within a loop) using strtok to remove the special characters like they are delimeters. It then adds the delimited word to the data field in L where it originally read from, and then iterates to the next dat field to restart the cycle.

qSort() function/Alphabetizing the List

In order to alphabetize an array of strings, the qSort() function will be needed.

Declaration:

qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))

1. The first parameter is the array to be sorted,
2. The second parameter is the number of elements from the array
3. The third parameter is the size in bytes of data type in array
4. Fourth parameter is a comparing function

qSort() function

Void used for comparing:

```c
/*for qsort() when ordering array alphabetically
more info on: https://iq.opengenus.org/qsort-in-c/
*/
int compare(const void* a, const void* b) {
    const char** str_a = (const char**)a;
    const char** str_b = (const char**)b;
    return strcmp(*str_a, *str_b);
}
```

strcmp return values are:

0 when equal, <0 when ASCII values are lower, >0 when ASCII values are higher

Frequency & Weight

To find Frequency & Weight we use "Strcpy" and "Strcmp".

Strcpy function serves to copy a set of strings to another set of strings

Strcmp function is used to compare two strings two strings, here we use strcmp to align the string and text files

1. We first iterate through the list of strings and compare a string to the next string in the list.
2. While we traverse the list, we hold the maximum frequency found to be used for weight later.
3. We iterate through this list one more time, but now we print each element that contains the word’s frequency to stdout.

Freeing Allocated Memory

The free() function allows is to deallocate the memory blocks which is previously allocated by the malloc function. Meaning, is helps free memory in the program that will have to be used later on.

References

String library: https://www.tutorialspoint.com/c_standard_library/string_h.htm

qSort(): https://www.tutorialspoint.com/c_standard_library/c_function_qsort.htm