CSCI 2270
2018 Instructor: Camilla Lambrocco
Project: Due Friday 7/27 11:59pm

It's all about space

What an exciting time, we are approaching the end of the semester!

In this final assignment we will see how we can apply the ADTs studied to real word problems. Have you ever wished you could download or stream movies faster? What about sending heavy documents via email? What if we could store massive amount of data and retrieve it really fast?
Well well.. I know what we can do, we can compress it! Not only we can compress it but we can store it in a convenient table and access it in almost constant time.
Read the write up carefully before starting. Throughout the writeup you will find some useful links.

IMPORTANT: read the files provided to you as they will tell you what to do. In the *HashTable.hpp and HashTable.cpp* files you will find "TODO" sections and questions. Respond to these questions right below the question and code accordingly.
The files *Utils.hpp and Utils.cpp* will have your utility functions as well as your Huffman encoding and decoding. These files have little information as I want you to do research on your own and be flexible in terms of helper functions. There are some stubs for structs and function that might be useful. You are supposed to write a short discussion about Huffman encoding and decoding algorithms you have to implement.

Description

The project is divided into three sections: Data Handling, Compression, and Storage. Your goal is to read in a file that has book excerpts, compress each excerpt and store a complex object called *zippedBookNode* into a HashTable. You are provided with the following files:

- *book_excerpt.txt*
- *Utils.hpp and Utils.cpp*
- *HashTable.hpp and HashTable.cpp*
- *FinalProject.cpp*

Read the .h files and implement the code necessary to complete the assignment/project. You are welcome to create your own helper functions. If you do so, please implement them in the Utils class (define your functions in the .h and accordingly implement them in the .cpp.)

Part 1: Data Handling

In this first section you have to handle the data extracted from a file by storing the title and the compressed body of a book into the struct *ZippedBookNode*. You will read data found in the file *book_excerpt.txt* (provided in Moodle).
(Note: your file should read in *book_excerpt.txt* as a command line argument.)

The structure of the file is the following:

```
#Forrest Gump
When I was born, my mama name me Forrest, cause of General Nathan Bedford Forrest who fought in the Civil War. Mama always
said we was kin to General Forrest's fambly someways. An he was a great man, she say, cept'n he started up the Ku Klux Klan
after the war was over an even my grandmama say they's a bunch of no-goods. Which I would tend to agree with, cause down
here, the Grand Exalted Pishposh, or whatever he calls hissef, he operate a gun store in town an once, when I was maybe
twelve year ole, I were walkin by there and lookin in the winder an he got a big hangman's noose strung up inside. When he
seen me watchin, he done thowed it around his own neck an jerk it up like he was hanged an let his tongue stick out an all
so's to scare me. I done run off and hid in a parkin lot behin some cars til somebody call the police an they come an take
me home to my mama. So whatever else ole General Forrest done, startin up that Klan thing was not a good idea – any idiot
could tell you that. Nonetheless, that's how I got my name.
**********
#Capital
In the history of primitive accumulation, all revolutions are epoch-making that act as levers for the capital class in
course of formation; but, above all, those moments when great masses of men are suddenly and forcibly torn from their means
of subsistence, and hurled as free and "unattached" proletarians on the labour-market. The expropriation of the
agricultural producer, of the peasant, from the soil, is the basis of the whole process. The history of this expropriation,
in different countries, assumes different aspects, and runs through its various phases in different orders of succession,
and at different periods. In England alone, which we take as our example, has it the classic form.
**********
#The madman
The madman jumped into their midst and pierced them with his eyes. "Whither is God?" he cried; "I will tell you. We have
killed him -- you and I. All of us are his murderers. But how did we do this? How could we drink up the sea? Who gave us
the sponge to wipe away the entire horizon? What were we doing when we unchained this earth from its sun? Whither is it
moving now? Whither are we moving? Away from all suns? Are we not plunging continually? Backward, sideward, forward, in all
directions? Is there still any up or down? Are we not straying, as through an infinite nothing? Do we not feel the breath
of empty space? Has it not become colder? Is not night continually closing in on us? Do we not need to light lanterns in
the morning? Do we hear nothing as yet of the noise of the gravediggers who are burying God? Do we smell nothing as yet of
the divine decomposition? Gods, too, decompose. God is dead. God remains dead. And we have killed him.
**********
#The Second Sex
Woman? Very simple, say those who like simple answers: She is a womb, an ovary; she is a female: this word is enough to
define her. From a man's mouth, the epithet "female" sounds like an insult; but he, not ashamed of his animality, is proud
to hear: "He 's a male!" The term "female" is pejorative not because it roots woman in nature but because it confines her
in her sex, and if this sex, even in an innocent animal, seems despicable and an enemy to man, it is obviously because of
the disquieting hostility woman triggers in him. Nevertheless, he wants to find a justification in biology for this
feeling.
```

Here you can see that # prepends the title of the book after which we append and an excerpt from the corresponding book. In the file there are several titles and excerpts. Each book is separated by ********** .
While you read through the file you will store every book' s title and compressed excerpt into respectively *zippedBookNode.title* and *zippedBookNode.c_excerpt*. In addition to these

two attributes you will store the root node of your Huffman tree (wiki lookup) in *zippedBookNode.huff_root*.

Part 2: Compression

In order for you to store the compressed excerpt and the root of the Huffman tree you will need to implement a specific kind of lossless compression called Huffman compression (Stanford handout.) This lossless compression follows a neat and intuitive algorithm. In order for you to succeed in the compression part you will have to use a priority queue. You are responsible for researching the algorithm and implementing it; here are some tips to get you started:

1) Compute the frequency of each letter in the excerpt of each book and extract the set (a set is without repetition) of letters of the excerpt.
2) Create a struct *huffNode* and let it be the node of your Huffman tree.
3) Each *huffNode* should store the letter and its frequency.

Example

Let's suppose that the body is the following:

i like skiing

and the node

huffNode myHuffNode;

then the properties mentioned above should return:

```
for i=0 to myHuffNode.size
        cout<<myHuffNode[i].freq << " ";

cout << endl;
for i=0 to myHuffNode.size
        cout<<myHuffNode[i].letter<< " ";
cout << endl;
```

Output:

```
4 1 2 1 1 1 1
i l k e s n g
```

You will have to write a decompression function as well. This will allow you to print out the book's excerpt.

Part 3: Storage

After you:

- successfully compress the body of the read in book using the Huffman encoding algorithm
- store the title of the book into *zippedBookNode.title*
- store the compressed body of the book into *zippedBookNode.c_excerpt*
- store the root of the huffman tree into *zippedBookNode.huff_root*

You should store every *ZippedBookNode* into a hashtable. The hashtable uses a hash function called *hashSum(...)* that takes the title of a book and returns the index where it should be stored in the hashtable. *hashSum(...)* is a method of the class *HashTable* (see .h for better understanding of the structure.) Unfortunately hash functions are imperfect which means that sooner or later there will be more than one entry hashed/mapped to the same value (index): this is called a collision. To resolve this situation you will implement a hash table using the concept of "chaining". In other words, when a collision happens between two or more books' title, you will chain the *ZippedBookNode* nodes together to form a linked list. This means that at the *n_th* index the hash table will store a pointer to the head node of the linked list that has our *ZippedBookNode*s.

Example (Part1,2,3)

```
// node of the singly linked list
struct ZippedBookNode {
      string title;
      double c_excerpt;
      *HuffNode huff_root;
      ZippedBookNode* next;

      ZippedBookNode(string name, double e, *HuffNode hr, ZippedBookNode* n)
      {
            title = name;
            c_excerpt = e;
            hr = huff_root;
            next = n;
      }
```

```
}

String book_excerpt = ...
…
…
…
int tableSize = 50; //the size of the table is an example
HashTable hashBookTable(tableSize) //
int index = hashBookTable.hashSum(“Forrest Gump”);



// create new node
ZippedBookNode* newZBookNode = new ZippedBookNode(
                              “Forrest Gump”,
                        encode(book_excerpt), // pseudo code
                        get_root(book_excerpt), //pseudo code
                        null)

ZippedBookNode* tmpZBookNode = hashBookTable.lookupTable[index]
// sanity checkssssss
…
…
// traverse list
while(tmpZBookNode->next)
      tmpZBookNode = tmpZBookNode->next;
// add new node at the end
tmpZBookNode->next = newZBookNode;
```

Program Structure

Display a menu

Once the your table is filled, your program should display a menu with the following options:

1. **Print Titles**
2. **Search Excerpt**
3. **Quit**

Use a command-line argument to handle the filename

Your program needs to open the *book_excerpt.txt* file using command-line arguments.

Menu Items and their functionality

Print Titles

Print all the titles stored in the hash table

Search Excerpt

If selected you should prompt the user for the excerpt's title they want to read. Here you will have to decode the compressed excerpt.

```
cout<<Enter title of the book<<endl;
```

Recap and submission details

What should your program do:

- Read in the file *book_excerpt.txt*
- Your code should be separated into 5 files *Utils.hpp, Utils.cpp, HashTable.hpp HashTable.cpp, and FinalProject.cpp*
- Separate definition and implementation in *.h* and *.cpp* files, respectively.
- Have the following functionalities:
    - Menu (5pts)
    - DataHandling (25pts)
    - Compression (15pts)
    - Storage (25pts)

Sumit a zipped file to the Project's submission link into moodle. You zip should have all the files mentioned above. Make sure to thoroughly comment your code and to write your steps and understanding on the Huffman encoding and decoding at the top your Utils.hpp file.

When submitting, please include all .hpp and .cpp files you used.

Appendix A

The following are pseudo snippets that should help you to format your cout statements and give you some insight on the functionality.

Print menu

```
cout << "======Main Menu======" << endl;
cout << "1. Print Titles" << endl;
cout << "2. Search Excerpt" << endl;
cout << "3. Quit" << endl;
```

Print Titles

```
for i=0 to lookupTable.end:
      //TODO get the value from the index
      cout<<"# "<<tmpBookNode->title;
      while tmpBookNode:
            //TODO traverse the list and retrieve the chained titles
            cout<<"->"<<tmpBookNode->title;
      cout<<endl;
```

Search Excerpt

```
cout<<Enter title of the book<<endl;
decompress(<title>);
```

Quit

```
cout << "Goodbye!" << endl;
```