CSCI 58000
Program 5 Overview

Due 12/06/23

Unlike previous programs, this will be a team effort. The teams are as follows (arbitrarily assigned):

Team 1	Bhimireddy, Gulia, Knight, Muthya, Toure
Team 2	Bi, Gupta, Kasa, Mettala, Mukthapuram Stapleton
Team 3	Clark, Gopal, Katta, Kulkarni M., Zhang
Team 4	Deshmukh, Isikbay, Kumari, Nair, Sholapurkar
Team 5	Gunturu, Khan, Manchikatla, Muddana, Richert
Team 6	Dasari, Deshpande, Kulkarni S., Nidumukkala, Pandit, Phirke

You can communicate via Canvas messaging or via regular email, whichever is more convenient.

There are 4 parts to Program 5, and on the deadline date of **12/06/23** you will need to turn in your code for each of the 4 parts. Obviously the 4 parts need to be done sequentially, that is, Part One first, etc. I have not set any intermediate deadlines, so it is up to you to set your own deadlines in order to meet the final deadline for all 4 parts. (*Hint*: generally speaking, each part is more difficult than the previous one, so you should waste little time in completing, say, Parts One and Two in order to have time to concentrate on Parts Three A and Three B and Parts Four A and Four B.)

Keep a Word "diary" of your team's efforts for all of Program 5 and the contributions of all individuals on the team. There are lots of tasks to go around - program design, implementation, testing, debugging, etc. All members of a team will get the same grade unless the diary says that someone made little or no contribution, in which case his/her grade will be some percentage of the team grade at my discretion.

The entire Program 5 is worth 65 points, and the points value of each part will be shown on the assignment for that part.

I strongly emphasize that you test your programs using Visual Studio on a Windows machine, otherwise you may run into difficulties when the programs are graded using Visual Studio.

Extra Credit: (10 points) Instead of turning in code for each of the parts to be executed separately, see if you can turn in just one main file that runs all the parts in the appropriate sequence in one execution.

Due 12/06/23 © 2023 Indianapolis, IN

CSCI 58000 Program 5
Part One

(10 pts) The first step in Huffman encoding of an archival text file is to do a character count. Write a program called *PartOne.cpp* to read a text file called *clear.txt* (which I will supply – this file you can use to test all parts of your Program 5 but your submitted work will be graded on a different text file). and write an output file *freq.txt*. The output file contains data on the characters in *clear.txt* and each character's non-zero count of occurrence in *clear.txt*, as in

```
,  7
B 17
e  35
f  2
```

etc. (Note that these are frequency counts, not percentages.) You can assume that *clear.txt* contains only characters from the standard ASCII 128 character set; order results in the output file by ASCII integer. The end-of-line character in a text file is a non-printable line feed character, with ASCII code 10. Write this character in the output file as LF. Be sure to include a prologue comment in your code with the team number and names of team members.

Hint #1: Include the *utility.h* file.

Hint #2: you need to read the text file character-by-character, so don't use the cin >> operation, which skips white space. Instead use the *get(x)* function, which is a function of the input stream. Since it is a text file, however, not a numeric file, you can use the *eof* function. Note that the *clear.txt* file you want to compact is supposedly very large (even though the *clear.txt* file I will give you is very small), so do not read the entire file into memory.

Hint #3: you might want to use the *static_cast <int>*( ) and *static_cast <char>*( ) typecasts. For example, static_cast <int>('a') results in 97, the ASCII code for the character 'a', and static_cast <char> (97) results in 'a' .)

Start a Word diary of your team's efforts for Part One of Program 5 and the contributions of all individuals on the team.

Due 12/05/23 © 2023 Indianapolis, IN

CSCI 58000 Program 5
Part Two

Based on Part One, you now can read a text file *clear.txt* and create a file *freq.txt* with the characters and frequency counts for *clear.txt*.

(15 pts) Use *freq.txt* to create the Huffman codes for the characters in the *clear.txt* file I posted. Your program (*PartTwo.cpp*, separate program from PartOne) should create a "code table" that gives each character and its binary Huffman code. Save this information in an external file called *codetable.txt* where each line of the file is a character and its code, ordered by the ASCII value of the character. (Presumably you will decode a file at some later date after you encode it, and you will need the codes to do the decoding, that's the reason for the *codetable.txt* file.) Be sure to include a prologue comment in your code with your team number and names of team members.

Hint #1: You should set up a binary node struct; see the slides for Chapter 12. You won't need a parent pointer, but you should store not only a character but also its frequency count from the *freq.txt* file.

Hint #2: For the list of frequencies in sorted order by frequency, you probably want to use either a linked list or a min-priority queue. Rather than write these container classes yourself, it's easier to use the Standard Template Library *list* class or *queue/priority queue* class. Check http://www.cplusplus.com/reference for functions available for use with these classes.

Hint #3: There may be duplicate frequencies as you build the frequency list, that is $f_i = f_j$. Depending on how the values are inserted into the sorted list or priority queue, f_i could appear before or after f_j. How this choice is made will affect the digits in the corresponding code strings, but the code lengths should be pretty much the same in either case.

Hint #4: Remember that arrays in C++ are automatically passed by reference without including the &. To get the effect of pass-by-value, use const in the parameter list.

Remember to update your Word diary of your team's efforts for this part of Program 5 and the contributions of each individual on the team.

Due 12/06/23

CSCI 58000 Program 5
Part Three

(20 pts)

Part Three A
Using the *clear.txt* and *codetable.txt* files, *PartThreeA.cpp* should read each character from *clear.txt*, find its corresponding binary code, and write this (**as a string**) to a text file called *coded.txt*. Again, do not bring the entire *clear.txt* file into memory.

Hint: When you look at the *coded.txt* file for the given *clear.txt* using Notepad, you should be able to see the code for "M" as the first few characters in the file, the code for "a" as the next few characters, etc. Of course, you could just wait and see what happens when you decode it!

Part Three B
Using the *coded.txt* and *codetable.txt* files, *PartThreeB.cpp* should read the characters in from *coded.txt* (again, even the *coded.txt* file could be large, so do not bring it into main memory) and decode them. (There's a brute-force way to do this and there's a more elegant way to do it. Either way is OK for this program.) Write the results to text file *decoded.txt*. Compare the content of files *clear.txt* and *decoded.txt*. They should be the same.

Add to your Word diary the contributions of each member of the team for Part Three.

Analysis: Compare the size of the original *clear.txt* file and the encoded file as follows: Get the number of bytes for *clear.txt* ("counting bytes" can be done within Windows File Explorer by right-clicking on the file name and looking at Properties). Do the same thing for *coded.txt*. Explain (in your Word diary file) why the *coded.txt* file is larger than the *clear.txt* file when we are supposed to be doing data compaction.

Due 12/06/23

CSCI 58000 Program 5
Part Four

(20 pts)

Part Four A: The file *coded.txt* created in Part Three is not a space-saver because we are writing out the code for a given character as a string of 0's and 1's, that is, as a string of characters. Think of this as wasting 7 out of every 8 bits.

So start over again by reading from *clear.txt* and finding the code for a given character, just as you did in part 3A. But this time you want the coded file to be the actual string of bits. For example, suppose *clear.txt* starts with ABC and the codes are

A: 00
B: 010
C: 100

Then the coded file should begin with 00010100, one byte of data. To accomplish this, your *PartFourA.cpp* will have to work at the "bit level". C++, due to its C heritage, provides mechanisms for doing this. You will need to keep an internal "byte" on which you manage the bits making up the byte. [I am using byte here as a generic term; you will have to select a data type to support this idea – I suggest you investigate the C++ **bitset class**. You will need #include <bitset> in order to use this class.] You'll also have to keep a marker as to which bit in the byte you are working on. When all the bits in the byte have been properly set, then write the byte to the binary file *codedalt.txt* (convert the byte to an ASCII integer, then write the corresponding character – that's one byte - to the file). Note that the code for a single cleartext character may spill off the end of one byte and into some of the bits of the next byte. The final byte in the file may contain extra garbage bits. If you try to read *codedalt.txt* in, say Notepad, it will just be a bunch of nonsense *because it is a binary file*.

Part Four B: Using the *codedalt.txt* and *codetable.txt* files, *PartFourB.cpp* should read "bytes" in from *codedalt.txt* (remember, you are reading from what kind of file?) and decode them back into characters (again you'll have to manipulate the bytes to get at the individual bits.) Write the results to text file *decodedalt.txt*. Compare the content of files *clear.txt* and *decodedalt.txt*. They should be the same (except perhaps for a bit of garbage at the end of *decodedalt.txt* left over from the final bits of the last byte in the file).

Analysis: In your Word diary, compare the size of the files *clear.txt* and *codedalt.txt*.
Add to your Word diary the contributions of each individual on the team for Part Four.

Submit via Canvas:

- Your program files (Parts One, Two, Three A and Three B, Four A and Four B) plus any other needed code files, including utility.h) – BUT SEE THE EXTRA CREDIT NOTE ON THE OVERVIEW PAGE.
- Your Word "diary"

DUE: Wednesday 12/06/23 by class time.