CS 235 Final

Spring 2013

Version 0.8

Instructor: R. P. Burton

June 19- 20, 2013 (Wednesday through Thursday)

due in the Lab on Thursday not later than 7:00 p.m.

Penalty for submitting the final late:

No Credit

Open Book (course text and your CS 142 course text only)

Open Notes (including your own Lab solutions)

Open Secondary Storage Device: yours only

Open Laptop: if you wish

Open Course Website (and the CS 142 course website) and the C++ API, but no other Internet resources (such as non API resources on the C++ website)

Closed Neighbor (and everyone is thy neighbor)

**\*Instructions\***

(Please read carefully)

1. This midterm consists of a C++ programming problem with optional extra credit. Read and understand the statement of the problem completely before beginning to design, code, and test. As part of your design, consider the test cases that will establish the correctness of your solution. Test your solution thoroughly before submitting it.
2. Produce a solution, which consists of your C++ code, with a comment at the beginning of each file (both .h and .cpp) which includes your name, your student ID number, and “CS 235 Spring 2013 Midterm.” Upload your completed project by compressing the files and submitting them through Gradebook with TA assistance. If a packet is not collected by a TA upon submission, your solution will not be graded and you will receive no credit for the exam. Attribute any code taken from or based on other sources (except for the course texts and the course websites). Attributed code copied from or based heavily on outside sources is worth half credit. Unattributed code copied from or based heavily on outside sources is worth no credit.
3. Understanding the problem correctly is part of the examination. If something seems unclear, ask a CS 235 TA for clarification. You may pose questions to the CS 235 TAs at any time. However, the TAs generally are not permitted to answer questions related to design, C++ implementation, debugging, or testing.
4. Prior to submitting your midterm, score it using the attached scoring sheet (this will help you maximize your points and will help us grade your exam accurately).
5. When you are finished, go to the course website and follow the link labeled “Submit Exam” in the Exam Menu.
6. Sign the Midterm Scoring Sheet (a) to certify that you have reviewed your CS 235 scores for the term and notified a TA in writing to any errors or omissions, (b) to request that your midterm be graded, and (c) to certify that no unfair information related to the midterm has been received by you, either directly or indirectly, and that none will be conveyed by you. If we discover that you cheated or assisted someone in cheating, intentionally or unintentionally (including accidentally), your score for this exam may (and probably will) be rand() % 0. We’re serious.

**I’ll Huff and I’ll Hash, and I’ll Blow You Some Bits**

|  |
| --- |
|  |

*Problem Statement*: Design, implement, and test a C++ program that:

* prompts the user for a file containing several characters. This file will be used to construct the Huffman tree with which subsequent encoding and decoding will be done. This will also be the message to be encoded/decoded
* identifies each character contained in the text and maintains a count of the frequency of occurrence of each character
* determines Huffman codes for each character
* encodes the original message
* decodes the original message
* compares the decoded message to the original message, and writes the results to an output file (send all output to \OUTPUT.txt)

*Requirement Notes*:

* Your encoding and decoding must be in accordance with the ‘Rules for Your Tree’ in a section included below.
* The comparison of the decoded message to the original message should include the input, the encoding, and the decoding, separated with at least one line in between each part. This should also include a statement reporting whether the original message and the decoded message are identical.
* Your program needs to be able to handle up to 100 different characters in any given file. These characters will be taken from standard 128 character ASCII. A minimum of 1 character will be provided to the program.

|  |
| --- |
| *Rules for Your Tree:*  For any given set of text, there could be multiple trees that are all optimal. This could lead to problems if two parties want to exchange messages that are encoded with trees generated using the same sample text. If each party generates a different, but still optimal, tree, they could end up with nonsensical decoded messages. To avoid this problem for this exam, use the following set of arbitrary choices that specify how certain aspects of the tree should be handled.   * + The left branch of a node will use a 0 in the encoding and the right branch use a 1.   + When combining two characters or nodes to make a new node in your tree, the character or node with the lower frequency is placed on the left and the character or node with the higher frequency is placed on the right.   + In situations where there are duplicate frequencies, a few rules need to be used to create the ordering of your priority queue. Original characters should come before combined nodes. When ordering two original characters, the ordering of the ASCII table should be used. When ordering two combined nodes, the older node should come first (**hint**: number each node as it is created). |

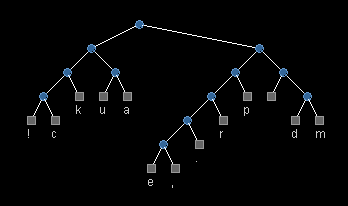
For your convenience when testing, we provide two strings: a message and the encoding of that message: using a tree generated from a sample text. The node that appears blank on the right side of the tree is for the space character. In the case of your program, the sample and the message will be the same string, although in the example below a separate string is used to generate the tree.

Sample Text: muckduck! muuuud draumak pmdap dp ucmr, paumeu dpapkam. mka. rpkuakdap pamk aprckpku.arp pruak, dcd ckap! r! p! .!mrp

Sample Message: ck rp, cdm, ku!eaa.

Encoding **-** 000100111010011011000011100001111011111000011100010100 00010000001101110001

Decoding - ck rp, cdm, ku!eaa.



Note: The node at ‘110’ represents the space character

.

**Extra Credit (15 Points):**

Uses hash tables to store the characters, their frequencies, and their Huffman codes. This hash table should be printed to a file before the program terminates (send all hash table output to \HASH\_OUTPUT.txt)

The hash table in the file should be in the following form:

Char Frequency Huffman Encoding

a 57 11011001

& 21 011011001001

**Midterm Scoring Sheet**

**Student Grading TA Grading**

\_\_\_/ 60 pts \_\_\_/ 60 pts – Make Encoding

\_\_\_/ 15 pts \_\_\_/ 15 pts – Correctly reads in data from a file

\_\_\_/ 25 pts \_\_\_/ 25 pts – Builds a Huffman tree based on file data and rules

\_\_\_/ 20 pts \_\_\_/ 20 pts – Implements a priority queue for tree creation

\_\_\_/ 30 pts \_\_\_/ 30 pts – Encode

\_\_\_/ 30 pts \_\_\_/ 30 pts – Encodes the message from the file into binary

\_\_\_/ 45 pts \_\_\_/ 45 pts – Decode

\_\_\_/ 5 pts \_\_\_/ 5 pts – Decodes the message from binary into text

\_\_\_/ 40 pts \_\_\_/ 40 pts – Uses the Huffman tree for the decoding

\_\_\_/ 15 pts \_\_\_/ 15 pts – Output

\_\_\_/ 15 pts \_\_\_/ 15 pts – Prints the messages into the \OUTPUT.txt file

Extra Credit

\_\_\_/ 15 pts \_\_\_/ 15 pts – Hash table use and printing to \HASH\_OUTPUT.txt file

\_\_\_/ 150 pts \_\_\_/ 150 pts – Total

Printed Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ BYU ID:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, 2013

(Signature) (Date)

\*signing here verifies that you have read and agree to all terms of the exam

**This section to be filled out by TAs**

Day Submitted: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Received by T.A. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

TA to Student Comments: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Graded By:**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_