CS 101 **Project 4** **Summer 2016**

This project implements k-ary Huffman coding, which extends the standard binary Huffman coding algorithm. You will write four separate programs (**count.cpp**, **build.cpp**, **encode.cpp**, and **decode.cpp**) as described below. It is sufficient to implement arities k=2 through 10 for full credit, and optionally you may also implement arities k=11 through 36 for extra credit.

The **count.cpp** program reads a source file and counts the number of appearances of each printable character (ASCII 32 through 126) that occurs in the text. It outputs a list of each such character's ASCII value and number of appearances (frequency), in ascending order by ASCII value. Do not output any control characters, or any character that does not occur within the text. Usage: ./count source.txt freq.txt

The **build.cpp** program reads the arity k (an integer from 2 to 36) and the frequency data generated by the count.cpp program. It creates a k-ary Huffman tree (see explanation below), and outputs a list of each character's ASCII value and k-ary Huffman code, in ascending lexicographical order of the Huffman code strings. It also outputs the nodes of the Huffman tree in the order visited during a preorder traversal. Leaf nodes are displayed as `L:xxx` where xxx is the ASCII value. Internal nodes are displayed as `I:xxx`, where each internal node is assigned a unique label xxx when it is created, starting with 256 and incrementing for each subsequent node. Also, to guarantee that the Huffman tree is unique in case of nodes having the same frequency, your heap should break ties by giving preference to nodes with smaller xxx values. Usage: ./build k freq.txt code.txt tree.txt

The k-ary Huffman tree generalizes the standard binary Huffman tree, such that each internal node has exactly k children. If n is the number of leaves, then a k-ary tree can exist only when (n–1) is a multiple of (k–1), so if this condition is not true initially, then add sufficiently many extra leaves each having frequency 0 to make the condition true. Assign each extra leaf a unique ASCII value, starting with 128 and incrementing for each subsequent leaf.

The k-ary Huffman code generalizes the standard binary Huffman code. It labels the child links using digits in base k. Each digit is represented as either a numeral (0 to 9) or a lowercase letter (a=10, b=11, c=12, …, z=35). Example: when k=16, use hexadecimal digits 0123456789abcdef.

The **encode.cpp** program reads the k-ary Huffman codes produced by tree.cpp, and the source file to be encoded. It encodes the source file by replacing each non-control character with its corresponding k-ary Huffman code, and then it outputs the encoded text file. Usage: ./encode code.txt source.txt result.txt

The **decode.cpp** program reads the arity k, the preorder traversal generated by the tree.cpp program, and the encoded source file to be decoded. It uses the preorder traversal to re-build the k-ary Huffman tree, and then it uses this Huffman tree to decode the encoded text file. Finally it outputs the decoded plain text, which should exactly match the original source file that was encoded. Usage: ./decode k tree.txt result.txt source.txt

Please carefully read the following requirements:

- You must do your own work; you must not share any code. If you violate this rule, you may receive an invitation to the dean's office to discuss the penalties for academic misconduct.
- Make sure your program runs properly on cs-intro.ua.edu. Your program will be graded on that system.
- Submit your project in a zipfile that contains your source files count.cpp, build.cpp, encode.cpp, and decode.cpp. There should be no subdirectories or extra files.
- If you violate the requirements such that it breaks our grading script, your project will be assessed a significant point deduction, and extreme or multiple violations may cause the project to be considered ungradable.
- A set of example input/output files and a test script are provided. Run the following commands to test your program. If any differences appear, then your project does not match the specifications. Points will be deducted for each incorrect line of output.

  ```
  unzip *.zip
  chmod u+x testscript.sh
  ./testscript.sh
  ```

- Alternatively, instead of running the test script, you can instead enter these commands.

  ```
  g++ count.cpp –Wall –lm –o count
  ./count source0.txt f0.txt
  diff  freq0.txt f0.txt

  g++ build.cpp –Wall –lm –o build
  ./build 8 freq0.txt c0.txt t0.txt
  diff  code0.txt c0.txt
  diff  tree0.txt t0.txt

  g++ encode.cpp –Wall –lm –o encode
  ./encode code0.txt source0.txt r0.txt
  diff  result0.txt r0.txt

  g++ decode.cpp –Wall –lm –o decode
  ./decode 8 tree0.txt result0.txt s0.txt
  diff  source0.txt s0.txt
  ```

- Submit your project on Blackboard by the due date (11:59pm Friday). There is a grace period of 24 hours (until 11:59pm Saturday). Projects submitted on Sunday will be assessed a late penalty of 5% per hour. No projects will be accepted after Sunday.
- Double-check your submission when you submit it. Errors discovered later cannot be fixed and resubmitted after the project is graded. Projects will not be re-graded unless an error is found in the grading script or in the input/output files used during grading.