# Project 3

**Huffman Coding**

Submission Deadline: **6:00 am** on Saturday, 30th July, 2016

## Project Description

Huffman encoding is an example of a lossless compression algorithm that works particularly well on text and, in fact, can be applied to any file. It can reduce the storage required by a third or half or even more in some situations. Hopefully, you will be impressed with this excellent algorithm and your ability to implement such a nifty tool! You have to write a program that allows the user to compress and decompress files using the standard Huffman algorithm for encoding and decoding.

## Overview of the Program Structure

### bstream.h/.cpp

A set of stream classes that you can use to read and write one bit at a time. You do not need to make any changes to these files, though if you're interested in the C++ streams libraries you might find the implementations interesting

Let's first start off with an easy module—the bstream module is already written for you and all you need to do is use it. The bstream module exports a set of streams classes (similar to the istream and ostream classes provided by C++) that allow you to read and write data one bit at a time. All of your favorite operations on file streams still work for the bstream classes; you can use getline, get, <<, put, >>, etc. with the bstream classes. Additionally, you have access to three new functions:

```
//For ibstream:
int readBit(); //Reads a single bit from the stream.
void rewind(); //Moves back to beginning of file to read again.
long size(); //Returns the number of bytes in the open file.
//For obstream:
void writeBit(int bit); //Writes a single bit to the stream
long size(); Returns the number of bytes in the open file.
```

The streams libraries in C++ are part of a class hierarchy. The classes istream and ostream define generic behavior that is common to all input and output streams, respectively, while the subclasses ifstream, istringstream, ofstream, and ostringstream allow you to read and write data to and from files on disk and strings in memory. Similarly, our ibstream and obstream classes are base classes with four subclasses: ifbstream, istringbstream, ofbstream, and ostringbstream, which let you do bit-level I/O to and from files and strings in memory. From your perspective, the functions you will be writing will always work on the more generic ibstream and obstream classes. This allows your code to be used for compression/decompression of files (for actual use) and of strings (for testing purposes). Test.cpp is provided as a part of the project to test the functionality of this class. You will not need to do any work on this module; you're simply a client of its classes.

## Program and Sample Output

You can use the 'wget' command to get the project directory.

```
wget http://www.cse.buffalo.edu/~tamaltan/resources/CSE250/Summer16/Project3.tar
```

Extract the directory.

Run the sample code 'huffmanSample' to check the output expected.

```
./huffmanSample
```

The program should run on Timberlake. If the executable permission is not set, execute the command:

```
chmod u+x huffmanSample
```

to make the program executable.

### test.cpp

A sample cpp file to show how bstream works.

## Your Task

### huffman.cpp

This is the only file where you need to do any modification.

Implement void runEncoding()and void runDecoding() functions.

`runEncoding()` function gets 3 file names from a user.

- *inputFileName* The file to be encoded
- *outputFileName* stores the name of the encoded file. This is the file to be written after encoding.
- *freqFileName* stores the freq of each character

`runDecoding()` function gets 3 file names from a user.

- *inEncodeFileName* The encoded file which we want to decode
- *outEncodeFileName* This is the file name used to save the result after decoding.
- *freqFileName* stores the freq of each character. This file was generated during encoding.

For compiling your code run the Makefile by just typing make and then pressing enter. You can test your code with the provided sample files, alice30.txt, sample.jpg, etc.

For this project, you just need to submit the huffman.cpp file.

Connect to timberlake server and type the following command:

```
submit_cse250 huffman.cpp
```

## Grading Details

- Frequency file correctly generated – 25 pts
- File correctly encoded – 25 pts
- File correctly decoded – 25 pts

  You should thoroughly check your code for encoding and decoding before submission.

## Acknowledgement

bstream.cpp and bstream.h files were taken from `http://stanford.edu/class/archive/cs/cs106b/cs106b.1136/assignments/Assignment6-linux.zip`

Few lines of the project description were copied verbatim from `http://stanford.edu/class/archive/cs/cs106b/cs106b.1136/handouts/230%20Assignment%206.pdf` You may wish to look at the pdf for more details, though it is not required.