Sheffield Hallam University

Faculty of Science, Technology and Arts

Department of Computing Project (Technical Computing) [55-604708] 2019/20

Author:	**Joseph Mitchell Jones**
Student ID:	**25025654**
Year Submitted:	**2020**
Supervisor:	**Simon Andrews**
Second Marker:	**Alessandro Di Nuovo**
Degree Course:	**BEng Software Engineering**
Title of Project:	**Investigating Data Compression Optimisation Techniques and Methods**

Confidentiality Required?

NO ✓

YES ☐

Contents

Abstract

This report follows the research, development and analysis of two proposed new approaches towards data compression and encoding. There is always research and improvement being done in the data compression industry, working on and adding to prior implementations to improve key aspects of them. In such an important area of computing, any slight improvements over previous methods can be incredibly valuable. This project seeks to identify an unexplored area of compression and encoding that could be improved, and produce and implementation to find any advantages or disadvantages between it and the original.

1 Introduction

The main goal of this project is to identify and implement a potential improvement to a data encoding and compression method. The implementation will be tested both with and without the improvement to produce fair results on the performance of both. The outcomes of these tests will determine the effectiveness of the improvements implemented and the success of the project overall. This project was chosen because there are still parts of the compression area that have not been fully explored, as well as use cases that may have improvements to be created.

1.1 Project Aims

The core aims of the project are:

- Identify and understand the current industry compression and encoding methods.
  An understanding of the current state of industry methods will be vital in the context of the project.

- Understand the use-cases, advantages and disadvantages of each of these methods.
  Gathering information on what each method is good or bad at will aid in finding a potential area for improvement that has not been explored.

- Identify and Implement a potential improvement to a current compression and encoding method.

An improvement will be implemented alongside a base version of the method to allow for fair testing to be carried out.

- Determine the advantages or disadvantages of the potential improvement over the base method.
  Experiments will be carried out to test the performance differences between the base implementation and the improvement.

1.2 What is an Encoding Method?

An encoding method is a method that seeks to modify the input data to create an output more convenient for compression. This may take the form of moving characters to create more repeating patterns or replacing data with smaller versions. An example of this would be Huffman coding, where more common characters are replaced with smaller bit codes with less common characters having longer bit codes. Encoding methods are often used before compression to aid in speed or compression ratio.

1.3 What is a Compression Method?

A compression method is a method that seeks to reduce the overall size of the input data, usually through representing repeating patterns as much smaller entries in the data. Most compression methods will use one of two methods: a dictionary-based method, in which data is added to a dictionary, then any later repeats referenced by index, or a pointer-based method, in which data is replaced by pointers to earlier points in the data stream.

1.4 Report Structure

Chapter 2 covers the research done into the current industry standards, including their use cases, advantages and disadvantages. Chapter 3 follows on from this research detailing proposed new approaches to be implemented. Chapter 4 covers methodologies used in the development of the project, this includes details such as the metrics and experiments designed to determine the success of the project as well as methods to aid in the development stage of the project. Chapter 5 includes the key necessary functionalities of the project, the logical structure, problems encountered and the process for resolving them. Chapter 6 covers the evaluation of the overall success of the project, comparing the metrics gathered from the improvements implemented to ascertain the value of the proposed improvement. Finally, chapter 7 will cover further discussion of results obtained in chapter 6, as well as possible uses for and any further improvements to the implementation.

2 Background Research

Lempel-Ziv Family

LZ77 (LZ1)

LZ77 (Ziv & Lempel, 1977) is the first of a series of data compression algorithms created by Abraham Lempel and Jacob Ziv in 1977. The core concept of LZ77 consists of a sliding window compression method, wherein data in a given range from the head is processed rather than through the use of a dictionary. Compression is achieved through replacing raw data with pointers to earlier in the data stream, allowing large sequences or data to be represented by a few numbers: Offset of the pointer, Length of data to be used from that offset, and the value of the following character. (Zeeh, 2003)

Example:

In this example, I will use the format of chunks denoted (x,y,z) where x is the distance (backwards offset), y is the length of the string to be replaced, and z is the following character after the replacement.

Input: aabbbcccdccdcc

Output: (0,0,a),(1,1,b),(1,2,c),(1,3,d),(3,4,c)

The above example shows both advantages and disadvantages of the LZ77 method. The compression is done as follows:

- The first chunk will always have a distance and length of 0, due to there being no previous data. This is then followed by the next char, in this case the first letter ‘a’.
- The next chunk takes a distance and length of 1, taking the previous char and copying it, then appending the next char ‘b’.
- The next chunk shows one of the advantages of this method: the ability to copy lengths greater than the distance provided. This is essentially a wrap-around, returning to the start of the pointer area when it reaches the end.
- To illustrate this point in particular, take the string “abcd”, followed by (4,6,e) as an example. The output for this would be “abcd**abcdab**e”. Note how it loops into itself in the copy.
- This process is repeated until the final chunk. In the final chunk, the final char should be noted explicitly, and not as part of a pointer. In the above example, (3,4,c) was used rather than (3,5,)

This method is not without its issues, as can be seen in the output length of the above example. In cases where there is not a large amount of repetition of long sequences, the distance, length, data triplet can be longer than the data it aims to compress. This data has been addressed by many, perhaps most notably in LZSS.

This replacement method, as well as the name “Sliding Window Compression” is derived from the three objects used to achieve this compression. These are the “Dictionary”, “Buffer” and “Cursor”. The cursor acts as the head of the algorithm, being the point at which data is input and processed. The dictionary consists of a set length cache of data previous to the cursor, to be looked at when finding repeats. The buffer is very similar to the dictionary, but for data after the cursor, to be looked at when finding repeat matches also.

LZSS (Lempel–Ziv–Storer–Szymanski)

LZSS (Storer & Szymanski, 1982) is an improvement on LZ77 created by James Storer and Thomas Szymanski in 1982. It maintains a similar technique to LZ77, with changes to how the triplet system of LZ77 is used by removing the need to explicitly state the raw value of the first non-matching character (Zeeh, 2003). It also uses a slightly more intelligent system regarding replacing small matches with pointers.

LZSS gets around the issue of replacing single characters with triplets by using a slightly modified version of the chunk system. Rather than (distance, length, next) a system is used of (pointer, distance, length) and (pointer, next). In this, “pointer” is a bit value of 0 or 1, representing whether that chunk is a pointer or raw data. The size at which a pointer is used over raw data varies, however a size of 3 seems to be commonly used.

This also increases efficiency, as it avoids the need to put the next char in every chunk. Not including this can result in longer matching sequences thus reducing the overall size after compression. “There's no big advantage for LZSS in terms of the compression rate comparing with other compressions, however its speeds of compression and decompression are very fast, therefore it's usually used on the occasions when speed comes first” (Yuan, 2011).

This in turn means that in larger files with larger non-consecutive, repeating patterns, this compression method will allow for larger matches, reducing the size of any unmatchable data.

Example:

In this example, I will use the format of chunks denoted (x,y,z) where x is the pointer Boolean, y is the distance (backwards offset) and z is the length of the string to be replaced *or* (x,c) where x is still the pointer Boolean, but c is the raw char. In this example 1 will denote a pointer and 0 will denote raw data. The size cut-off for the use of pointers will be 2.

Input: aabbbcccdccdcc

Output: (0,a),(0,a),(0,b),(1,1,2),(0,c),(1,1,2),(0,d),(1,3,5)

While the above example is very short, it can still be seen how this may be more efficient with larger data-sets.

The process is as follows:

- No match can be found for the first char, ‘a’, in the history as it is empty, so raw data is output.
- No match of size 2 or above can be found for the next chars in the input, so raw data is output.
- No match is found for ‘b’ in the history, so raw data is output.
- A match of length 2 is found for ‘bb’, so a pointer is made to the first ‘b’, copying it twice.
- No match is found for ‘c’ in the history, so raw data is output.
- A match of length 2 is found for ‘cc’, so a pointer is made to the first ‘c’, copying it twice.
- No match is found for ‘d’ in the history, so raw data is output.
- A match of length 5 is found for ‘ccdcc’ so a pointer is made to the beginning of the previous 3 chars that make up ‘ccd’. A length of 5 is then given so it recurs to the first 2 ‘c’ chars.

LZ78 (LZ2)

LZ78 (Lempel & Ziv, 1978) is the second major data compression algorithm developed by Abraham Lempel and Jacob Ziv in 1978. It applies a similar compression method to LZ77, with the key difference being the use of a dictionary over a sliding scale method. This helps to avoid issues with sliding scale such as range of matching, but in turn introduces other issues such as handling dictionary growth.

LZ78 uses a key-pair method to encode data with a dictionary. The way this works is by showing two numbers: a position in the dictionary, and the following raw char. In this method, an entry can be seen as (position, value) where position can be either 0, indicating a value not yet in the dictionary, or a positive integer, showing that value's position in the dictionary. This is then followed by the first non-matching character, much the same as in LZ77. This entire sequence, including the first non-matching character, is then inserted into the dictionary.

Example:

Input: aabbbcccdccdcce

Output: (0,a),(1,b),(0,b),(3,c),(0,c),(5,d),(5,c),(0,d),(7,e)

Dictionary:

Dictionary Position	Value
1	a
2	ab
3	b
4	bc
5	c
6	cd
7	cc
8	d
9	cce

The process is as follows:

- 'a' cannot be found in the dictionary, so it is given a position 0 and then input into the dictionary at position 1
- The next 'a' is found in the dictionary, so is referenced with position 1. This is then input into the dictionary along with the next char, 'b' at position 2
- 'b' cannot be found in the dictionary, so it is given a position 0 and then input into the dictionary at position 3
- This then continues in this fashion until all data is compressed.

This data can then be decompressed by rebuilding the dictionary from the compressed data, using much the same method as was used to compress the data.

LZ78 runs into the same issue as LZ77 in the use of 'first non-matching character', but avoids many of the other issues by using a dictionary method over sliding window. This, however, has its own issues, most notably being the lack of dictionary management. This means that in a sufficiently large dataset, there is nothing preventing the dictionary from growing to large sizes.

LZW (Lempel-Ziv-Welch)

LZW (Welch, 1984) applies a similar approach to LZ78 as LZSS applies to LZ77, in removing the need to explicitly state the first non-matching character. It achieves this by first filling a dictionary with all characters in the input, then following the same dictionary generation and compression method as LZ78, but without the first non-matching characters (Blelloch, 2013). This means that the output of LZW consists purely of dictionary references, which in turn allows for longer pair matching and smaller sizes of output. This can, however result in a significant dictionary size, wherein "The size of the dictionary may grow so quickly that an effective method of maintaining the dictionary would be essential" (Pu, 2005).

Encoding Methods

Burrows-Wheeler Transform (BWT)

BWT (Burrows & Wheeler, 1994) is a method of rearranging strings of characters in a way the optimises the number of repeating patterns in a row, allowing for easier compression. The key aspect of this method is that it is entirely self-reversible, meaning the only requirement is additional computing on top of any other compression methods. This is achieved by finding each permutation of the input string, then sorting these permutations lexicographically, giving the optimally sorted string as an output along with an index referencing the original string.

Example:

Input: mercedes

Permutations:

0	m	e	r	c	e	d	e	s
1	s	m	e	r	c	e	d	e
2	e	s	m	e	r	c	e	d
3	d	e	s	m	e	r	c	e
4	e	d	e	s	m	e	r	c
5	c	e	d	e	s	m	e	r
6	r	c	e	d	e	s	m	e
7	e	r	c	e	d	e	s	m

Sorted:

0	c	e	d	e	s	m	e	r
1	d	e	s	m	e	r	c	e
2	e	d	e	s	m	e	r	c
3	e	r	c	e	d	e	s	m
4	e	s	m	e	r	c	e	d
5	m	e	r	c	e	d	e	s
6	r	c	e	d	e	s	m	e
7	s	m	e	r	c	e	d	e

Output: (“recmdsee”, 5)

In this output, “recmdsee” is the sorted string, and 5 is the index in the sorted table at which the original string can be found. This is all that is required to reconstruct the original string.

Uncompressing

The output string, from here referred to as ‘L’, is sorted lexicographically to give the first column from the original table, which will be referred to as ‘F’. This gives us:

L=“recmdsee”
F=“cdeeemrs”

As can be seen above, due to the fact that L and F were consecutive in the sorted table, each character in the same position of each string will have been consecutive also, with the character from L coming directly before the character from F. This is seen in the above example with “rc”, “ed”, “ce” etc.

To achieve full decompression, **column** L is essentially added to the table, then each **row** sorted lexicographically. This process is repeated N times, where N is the length of the string.

For example:

Add > sort #1:

0	r		c
1	e		d
2	c		e
3	m	>	e
4	d		e
5	s		m
6	e		r
7	e		s

Add > sort #2:

0	r	c		c	e
1	e	d		d	e
2	c	e		e	d
3	m	e	>	e	r
4	d	e		e	s
5	s	m		m	e
6	e	r		r	c
7	e	s		s	m

Add > sort #3:

0	r	c	e		c	e	d
1	e	d	e		d	e	s
2	c	e	d		e	d	e
3	m	e	r	>	e	r	c
4	d	e	s		e	s	m
5	s	m	e		m	e	r
6	e	r	c		r	c	e
7	e	s	m		s	m	e

As can be seen in the table above, the original sorted table from the compression is beginning to form from this repeated process. This adding and sorting would be repeated until it has been done eight times, recreating the entire original table. The index supplied with the sorted string would then be used to look up the row in question on the recreated table, giving the original, unsorted string. It is also possible to avoid the use of an index when compressing the string with the use of "start of string" and "end of string" characters. In this case the final string selected from the table would simply be the one starting and ending with these characters. This lexicographic sorting results in all information on every permutation and substring of the input data, in a size no larger than the original string (Fenwick, 2007).

The transformed string becomes easier compress due to common sequences of characters being moved near to each other. To illustrate this, take an input where the word "the" is used. (Burrows & Wheeler, 1994) state "When the list of rotations of the input is sorted, all the rotations starting with 'he' will sort together; a large proportion of them are likely to end in 't'. One region of the string L will therefore contain a disproportionately large number of 't' characters, intermingled with other characters that can proceed 'he' in English, such as space, 's', 'T', and 'S'".

It is worth noting that effective compression can only be achieved with large inputs, in the range of a few kilobytes of characters (Burrows & Wheeler, 1994). While this encoding method does not directly compress data, it is very useful when passing this data into other compression methods, such as LZ. This is due to the sorting of these characters into repeating sequences allows for faster compression and more reductions in size due to more consistent repeating patterns.

Move-to-Front Transform (MTF)

MTF (Ryabko, 1980) is a data encoding method designed to reduce the size of commonly used characters in a string by replacing those characters with an index in a stack of recently used characters. It achieves this by starting with predefined dictionary, for example the alphabet, and shifting any character encountered to the front of this dictionary after encoding it. This means that in a situation where there is a set of repeating characters, any after the first will be replaced by the index 0. This also results in less frequently used characters being pushed towards the back of the dictionary, with more frequently used ones remaining near the front.

For the following example, the dictionary used will be the standard English alphabet, where ‘a’ has index 0 and ‘z’ has index 25.

Example:

Input: aladdin

Output: 0,11,1,4,0,9,13

The process for the above is as follows:

Starting dictionary: abcdefg…

- ‘a’ is already at 0, so nothing changes
- ‘l’ is at position 11, so 11 is encoded and ‘l’ moved to the front

New dictionary: labcdef…

- ‘a’ is now at position 1, so 1 is encoded and ‘a’ moved back to the front

New dictionary: albcdef…

This then continues until the entire string is encoded. This method is also self-reversible as long as a standard starting dictionary is used.

Huffman Coding

Huffman coding (Huffman, 1952) is a method of assigning characters in a string different length bit codes based on how often they appear in the input string (their probability). This is achieved by creating a binary tree with a node representing each character and its probability, which is then used to assign codes. From a list of all nodes and their probabilities, the tree is constructed as follows (Blelloch, 2013):

- The two lowest probability nodes are removed from the list
- A new node is created as a parent node to these two nodes, with a probability equal to the sum of the previous two nodes’
- This new node is added to the list

This process is then repeated until only one node remains, which becomes the root of the tree.

Example

Input: alleless

Output: 111 0 0 110 0 110 10 10

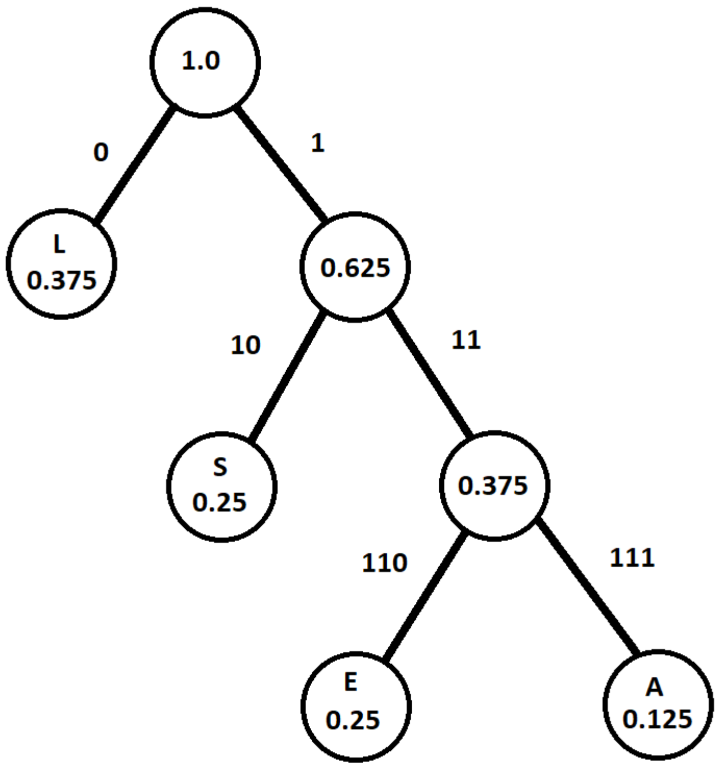

Fig 1.   The binary tree for the input “alleless”

Fig 1 shows the binary tree created from the input string “alleless”, with the numbers on the branches representing the bit code, and the numbers inside the nodes representing their respective probabilities. Please note that spaces have been added to the above output to make it more easily comparable to the tree. The key feature of this output code is that each code will have a unique prefix. As can be seen above, none of the codes can be mistaken for another. For example, 1111100 must always be read as AEL (111 110 0), since there is no code corresponding to 1 or 11. It is due to this unique prefix attribute that Huffman codes can be of variable length, as no delimiters or recording of code length is required, they can just be read bit by bit and decoded as soon as the input stream matches a code.

The number in each node refers to the probability of that node if it is a leaf node, or the sum of all of its children nodes if not. It can be seen here how the codes are assigned to characters, traversing the tree until a leaf node is reached, adding a ‘0’ to the code for each left branch taken, and adding a ‘1’ to the code for each right branch taken.

Prediction by Partial Matching (PPM)

PPM (Cleary & Witten, 1984) is a method of assigning more efficient probabilities to characters in a string, to allow them to be more effectively encoded. This is achieved by building up a contextual dictionary, showing probabilities based on previously seen characters. For example, in a given string the character ‘e’ may have a probability of 0.2. If, however, the two previous characters are ‘qu’, ‘e’ will in turn have a significantly higher probability in the region of 0.9.

3 Proposed New Approaches

From the research conducted prior, two new approaches have been identified as potential improvements to be implemented.

The first of these is a new approach to Huffman Coding (Huffman, 1952). Huffman Coding has always used eight-bit (one byte) chunks when encoding data. This is efficient in raw binary files and other data sources where the data is not expected to have a pattern, as each byte will be encoded individually, and repeating patterns of multiple bytes are unlikely to appear frequently. This may not always be the case, however, such as in written language processing, where multiple consecutive bytes can be more expected to repeat ('the', 'and' etc.). As such, the proposed new approach for this area is an implementation of Huffman Coding using variable bit lengths for encoding (sixteen-bit, thirty-two-bit etc.), as this will allow investigation of its performance on a variety of data inputs.

The second proposed new approach is a modification to the LZ78 dictionary-based compression method. In particular, the base version to be modified will be LZW (Welch, 1984). An issue that dictionary-based compression methods have to deal with is the handling of the dictionary when it becomes too large, for which multiple solutions are used. Of the current methods for handling this issue, none are particularly intelligent, with the most solutions being to destroy then rebuild the dictionary from scratch or make the dictionary static and continue. As such, the proposed new approach for this would modify the dictionary handling method, to use a weighting system on entries. This weighting would be based on the length of the entry, as well as the frequency with which it has been used. This will result in large entries that are rarely used being deleted from the dictionary first, with smaller entries used frequently being kept.

4 Methodologies

4.1 Testing Strategy

The goal of this project is to attempt to implement an improvement of an existing compression and encoding method, and as such, it is extremely important that the tests run to determine the effectiveness of the improvement are fair. This in turn will ensure that the results of the tests are reliable and give a valid overview on the advantages and disadvantages of the improvement. In order to ensure that these experiments are carried out properly, two versions of each method will be implemented. The first of these will be a base version of the implementation, without the proposed improvement. This will serve as the control to compare the implemented improvement to, ensuring that the only difference in the code being compared is the improvement in question. The second version to be implemented will be the version with the proposed improvement.

4.1.1 Metrics Used

The metrics used in evaluating the improvements will differ for each improvement, as they each aim to improve in different areas. The metrics to be used for the Huffman Coding improvement will be speed (how quickly the file can be compressed and decompressed), and compression ratio (the difference in file size between the uncompressed and compressed file). These values of these metrics will be measured and compared to reach a conclusion on the success of the improvement.

Compression ratio is the simplest of all the metrics to measure, as the file size of the input file can be measured, then simply compared to the file size of the compressed output file. Calculating the compression ratio from there is a basic division.

Speed is slightly more complex to measure. In order to achieve an accurate measurement of speed, a timer will be implemented in both methods, which will run for the duration of the processing time. This will ensure a specific time, down to the millisecond, is given for each run of the method. As to also ensure the reliability of the speed results, each test for the speed will be run on the same machine, with minimal other processes running. These tests will be run many times for each test data, allowing for the exclusion of any extreme outliers, and an average time to be calculated.

Slightly different metrics will be used in the testing of the dictionary improvement. As this code will not affect the output of the compression itself, it should not have any effect on the compression ratio whatsoever, and as such this metric is unnecessary for this implementation. The metrics used for the dictionary improvement are speed, which will be tested in the same way as outlined above, and memory usage.

Memory usage will be tracked by use of the tool Performance Monitor (Microsoft, 2013). This tool allows the logging of CPU and memory usage for a particular process. This will give an exact display of how much memory the implementation is using at any given time during its execution, and will graph the usage over time during execution.

4.2 Tools Used

This project will make use of the version control system git (The Git Project, n.d.) through the online tool GitHub (GitHub, n.d.). Using this system allows for the tracking of code changes, and provides a convenient central location from which the code can be retrieved across multiple devices. The change tracking functionality also allows for easy bug tracing, looking back through the code to find where it was introduced, and reverting the code to a previous state, effectively serving as a backup of previous versions. Git was selected over other tools, such as Sub Version (SVN), due to its wide adoption and status as the industry standard. Git also does not require any self-hosted infrastructure, and access to GitHub allows for use of features such as continuous integration (CI) and continuous development (CD).

Git also allows for the use of branches – additional named versions of the codebase, which can be pushed to and pulled from independently of any other branches. This is highly useful for this project, as branches can be used for testing features to be added, as well as for the implementation of the improvements. Once the base version of a method has been implemented, a branch can be created from that changeset, and the improvement created from that. This is convenient for having both of the implementations tracked and accessible from one central location.

When selecting the best environment for the development of the improvements, the environment that was picked was the Microsoft Visual Studio IDE (Microsoft, n.d.). This was decided on, as Visual Studio's debugging and analysis tools are extremely useful during the development process, making it significantly easier to root out any bugs that may appear and identify any areas that could be improved or made more efficient.

SCRUM (Scrum, n.d.) will be used throughout the development process to aid in the creation of the project. SCRUM has been chosen as the methodology to follow as it is agile, and allows for quick reactions and changes to any external factors. This is very likely to happen, as other work and assignments come in and priorities need to be shifted. It is this flexibility to unforeseen influences that makes SCRUM ideal for a project like this. The fast-paced nature of SCRUM also makes it an appealing methodology to follow, allowing for quick results and rapid prototyping, while building a level of expected pace for the project as development continues. Having frequent reflections will allow chance to improve in ability to perform and make any changes needed to the process to improve development (Vanderjack, 2015). SCRUM was chosen over other methodologies such as the Waterfall Model as it allows for much more flexibility during development. With waterfall, rearranging goals and dealing with unanticipated interference can be very difficult due to its rigid nature, and since additional work will be guaranteed to come in over the course of the development process, this flexibility will be vital for the project's success.

4.3 Programming Language Used

The chosen programming language for this project is C++, as it has many features that will aid in the development process and is a language the developer has experience with. In particular, it was decided that an Object-Orientated language would be used to aid in the creation of the Huffman Coding implementation, as custom classes would need to be utilised in the development process. Due to the nature of this project in processing potentially large files and datasets, the efficiency of C++ was deemed to be very important, as testing would rely on the program running efficiently when processing these large datasets. The access to pointers will also serve useful in the development of the improvements, in order to efficiently store and access data.

C++ is designed around this idea of efficiency from the ground up through principles such as "What you don't use, you don't pay for (in time or space) and further: What you do use, you couldn't hand code any better." (Standard C++ Foundation, n.d.). Further, when compared, "Java programs are typically much slower than programs written in C or C++. They also consume much more memory" (Prechelt, 1999).

5 Implementation

5.1 Huffman Coding Implementation

The development process for the Huffman Coding implementation consisted of multiple pieces of functionality. The nature of Huffman Coding meant that each of these steps is required for the following to be implemented, and as such each of the pieces of functionality are prototyped and implemented sequentially, over the course of multiple sprints.

5.1.1 Binary Tree

The first piece of functionality to be implemented was the binary tree, as this is necessary for all subsequent steps. The requirements for this step were:

- Processing an input file and generating a frequency map from it
- Creating a valid node from the data generated from the file
- Combining multiple nodes into one
- Creating a valid binary tree structure from these nodes

The first of these requirements is achieved by reading the file into a vector structure, then creating a map with a key of the data and a value of its frequency. This can then be passed into later functions to generate the binary tree.

At the core of Huffman Coding is the binary tree, and in order to assign codes to data pieces, one must be constructed. This structure is achieved through the use of two classes: 'node' and 'nodeHeap'.

The class 'node' simply acts as a node in a binary tree, containing that node's char, if it has one, the overall frequency probability of the node, and pointers to any child nodes it may have. While this is key to assigning codes and is what will become the binary tree in the end, on its own it does not achieve anything, resulting in the need for the second class: 'nodeHeap'.

The class 'nodeHeap' provides all functions necessary to generate and initialise the binary tree itself, as well as providing one object that can be operated on to complete this entire process, eliminating any potential crossover of functionality with other parts of the system. This consists of creating nodes from the initial item-frequency map and storing these. From there, additional functions allow the removal and creation of new nodes as children to other nodes, thus creating the binary tree.

The implementation of these two classes allows the 'buildTree' function to simply create a 'nodeHeap' instance, utilise the functions to combine nodes together and repeat this until only one node remains in the heap. This node is the root node that is used in assigning codes later on.

5.1.2 Assigning Codes

Once the binary tree creation was implemented, the next step was to implement a method for assigning codes to data pieces. The requirements for this step were:

- Navigate the binary tree from a root node
- Assign codes based on a node's position in the tree

Assigning codes to data pieces was the next step in the implementation of Huffman Coding, as this is the key part of the process, and the step that actually creates the compressed file.

In order to assign codes to each data piece, the binary tree is traversed until it reaches a 'leaf node' (a node with no child nodes). As the tree is traversed, for each branch taken, a number is added to the code. Whenever a left node is taken, a 0 is added to the code, and whenever a right node is taken, a 1 is added. This process continues until a leaf node is reached, at which point the code in its current state is applied to the data piece in that leaf node.

This process is achieved in the function 'displayCodes'. This function takes input in the form of: the root node of the binary tree, a vector to store the code in, and a map to store a code-data pair. The root node is then recurred upon, with the function being called from within itself every time a child node of the current node is found. This then continues until a leaf node is found, where the function outputs the code then returns. Due to the check for a right node being directly after the check for a left node, this results in the function effectively ‘going back up a layer’ and checking the right branch whenever a code is output and the function returns.

The result of this process is each leaf node of the binary tree being checked and assigned a uniquely prefixed code. It is because of this unique prefix that delimiters will not be needed in the final compressed output and the codes can simply be directly output on their own.

5.1.3 Transmitting Data

In order for the encoded data to be capable of being decompressed, some data must be passed along with the Huffman codes themselves, allowing them to be processed. Two methods of transmitting data to enable decompression were identified as possible candidates for this project.

The first of these methods was to transmit the initial dictionary of bytes and frequencies used to create the binary tree. This had the advantage of being simple to output and read in, as well as allowing for quick reconstruction of the binary tree at the decompression end.

The second method identified was to encode the binary tree itself, and output that along with the codes. The key advantage for this method was the potential to create a smaller output size than the above-mentioned frequency output method in larger files. The disadvantages of this method, however, include a significantly more complex encoding process, both for outputting the tree to the file and for reading it in from the file.

For the testing of the implementations, smaller files will be used in order to make faster results, as well as make the output files themselves more readable. It is because of this that the first method was chosen for the implementation of Huffman Coding, due to the advantages of the binary tree encoding method and the disadvantages of the first method being relatively insignificant at smaller file sizes.

5.1.4 Decoding Data

In order to decompress the data, the binary tree must first be rebuilt. In this implementation, this is done in much the same way it is during the initial compression; creating nodes based on the byte and its frequency, then merging these nodes to create parent nodes until only one root node is left.

The input stream is then read starting from the root node. For each ‘0’ encountered in the stream, the current node is set to its left child node, and for each ‘1’ is set to its right. This process is repeated until a leaf node (one with no children) is reached, at which point the byte stored at that node is output. The current node is then reset to the root and this entire process continued until the input stream has been read in its entirety.

5.2 LZ Dictionary Implementation

The development process for the LZ dictionary implementation consisted of two initial development phases, followed by the implementation of the improvements to the method. The first of these phases was the creation of the compression and getting a base layer of functionality. This was then followed by a similar process for the decompression functionality. These two parts took a large part of the development time, as they needed to be functional for the subsequent stages to work. The improvement implementation stage followed a process of rapid prototyping, in order to find the most effective way of implementing the weighting system.

5.2.1 Dictionary Creation

The creation of the dictionary is achieved in two steps: the initialisation of the dictionary, and the addition of entries to it.

For the initialisation process, the dictionary is created and filled with 256 entries, one for each possible permutation of a byte. The use of this initialisation technique means that as long as the data is processed in a byte by byte fashion, no details of the initial dictionary will need to be transmitted along with the compressed data, due to the starting logic being the same across compression and decompression.

From there, the input stream is processed, outputting indexes to pre-existing entries in the database, appending previously unencountered character combinations to the dictionary. This will result in a dictionary that contains each consecutive character combination that appears in the input stream, and an output that consists entirely of indexes of that dictionary.

5.2.2 Weighting Entries

In order to selectively remove entries from the dictionary when its maximum size is reached, a weighting system was implemented as an improvement. To achieve this, each entry in the dictionary had its use frequency tracked, increasing every time it is used in the compression or decompression. When the dictionary then hits its maximum size, a weighting is calculated from the frequency of use and the length of the entry in question, with a long entry that has rarely been used being more likely to be removed.

5.2.3 Pruning Entries

In the base implementation, pruning is carried out simply by resetting the dictionary. This is done by reinitialising it - removing all entries then populating it with the original 256 entries.

For the improvement implementation, the weighting outlined above is utilised. Whenever the dictionary reaches its maximum size, the weights of the entries will be compared and the entry with the lowest weight (a longer entry that has been rarely used) will be removed from the dictionary to make room for the next.

5.2.4 Decompressing Data

Due to how the dictionary is initialised, no details need to be transferred from compression to decompression in addition to the list of indexes. The same rule sets for creating dictionary entries and pruning them is applied to decompression of the compressed file.

The first entry is guaranteed to be a single byte due to the method of compression, and since all permutations of a byte exist in the initialised dictionary, the first entry is guaranteed to exist in the dictionary and can therefore be decoded and output. From this starting point, the rest of the input stream can be read and entries added to the dictionary in much the same way. As long as the dictionary entries are created and pruned in the same fashion using the same set of rules, the indexes for each of the entries will remain consistent between compression and decompression.

This process is repeated until all of the data has been read and output, resulting in a fully decompressed file.

5.3 Issues Encountered During Development

During the development process many issues occurred, both expected and unexpected. The first major issue was one that had been expected to take some time to implement, in the method for traversing the binary tree created in the Huffman Coding implementation. This feature was vital and needed to be done in a couple of different ways for compression and decompression, as it was the most vital part of both assigning and decoding the created Huffman codes. The initial method implemented - beginning from the root node and setting it equal to one of its children in order to traverse down – worked for the decoding phase of the program, but unfortunately did not for the encoding phase. In order to resolve this, many methods of traversing the tree were rapidly prototyped until a solution was arrived at with the recursion-based method implemented in the final version of the project. While this issue was anticipated, it took a larger amount of development time than expected, and caused a slight delay in the project. Despite this, it did not prove to negatively impact the implementation’s quality.

In order to decode the codes created by the Huffman Coding implementation, the initial dictionary needed to be output for use in the decoding stage. Due to the nature of the data being read in, making the file easily readable by the decoder was a challenge to resolve. Using standard delimiters and writing entries to file was not possible, as any character could be written, and thus delimiters would not be reliable. This issue was resolved through the use of numeric values for each byte written, allowing for use of delimiters to read in the values stored. This could have potentially been avoided through the use of the binary tree writing method, in which the entire binary tree would be directly written to file, instead of the dictionary writing method that was used.

During the development of the LZ dictionary, it was found that the dictionary would occasionally ‘skip’ entries when building. For example, the string ‘ear’ may be added when the string ‘ea’ did not yet exist in the dictionary. This caused issues when decompressing, as subsets of one entry may not exist to be decoded. In order to resolve this, logic was added to ensure that any large entries were added byte by byte, ensuring that there would be no entries added that were too large. While this unanticipated issue did cause a delay in the project, it was quickly resolved and did not negatively affect the project’s quality.

The largest issue encountered during development was an inconsistent output in large files while using the weighted version of the LZ dictionary implementation. In large files, when using the weighting system, the output towards the end of the file would sometimes become corrupted, outputting additional bytes where there should be none, sometimes outputting different bytes altogether. Unfortunately, this issue was only discovered very late into development, and as such was unable to be resolved.

This resulted in a huge amount of time being wasted debugging and prototyping new methods for the weighted removal system. While it was determined that this issue occurs during the pruning of entries from the dictionary and is believed to be an issue in handling entries in the middle of the dictionary with equal weights, the nature of the issue being inconsistent and time-intensive to test resulted in it not being resolved. This issue could potentially have been caught earlier with a stricter testing strategy, checking the data whenever it is input, output or modified. Given opportunity to study further into this area, this is something I would seek to resolve.

While this is a major issue, the only area that is affected by this flaw is the file size of the decompressed file. Since this was not one of the variables being measured in the experiments for the LZ dictionary implementation, the data gathered from these experiments will still be valid and useful.

6 Evaluation

To determine effectiveness of the improvements implemented, an evaluation took place to analyse the advantages and disadvantages of each improvement, as well as the overall quality of the implementation. Critical analysis of the project as a whole was also performed, addressing areas of the project that did not go well and providing insight as to how they may be improved in future, or how they may change if the project were to be done again.

6.1 Analysis of Huffman Implementation

The metrics analysed for the Huffman implementation were speed (how quickly compression and decompression could be achieved) and compression ratio (how much of a reduction in file size was achieved from compression). In order to test this, several randomly generated files of varying sizes, as well as one written language text file, created using "Lorem Ipsum" (Microsoft, 2018), were compressed and decompressed using both the base version and the improvement The compression time, decompression time and the difference in size between input and output were all recorded for each file and used to assess this implementation.

6.1.1 Speed Comparison

Input File	**Time Taken (Base / 8-bit) (s)**	**Time Taken (Improvement / 32-bit) (s)**
Lorem Ipsum (100KB)	0.808	1.556
Random (100KB)	0.917	34.215

Random (200KB)	1.707	140.746
Random (300KB)	2.602	343.199
Random (5MB)	40.129	>45m
Random (10MB)	81.919	>45m

The time taken for each file was recorded programmatically using a timing circuit to record the time taken for execution. This code was adapted from an earlier assignment for Effective C++, with permission from Paul Parry.

As can be seen from the results above, the base 8-bit version was significantly faster for both compression and decompression than the 32-bit implementation. As the file size increased, the time taken for the 32-bit implementation increased exponentially, becoming extremely slow after a relatively small increase in file size. Due to this, timings were not obtained for the randomly generated 5MB and 10MB files, as these both took longer than 45 minutes to complete, at which time they were terminated.

The pattern of increase can still be clearly seen from the results obtained, with the 32-bit implementation suffering massively from the increase in file size. The vast majority of the computation time lost was in the creation of the binary tree, as this is the most intensive area. This is most likely due to the significantly higher amount of potential Huffman codes in the 32-bit implementation versus 8-bit. For comparison, there are 256 possible Huffman codes in an 8-bit implementation, and 4,294,967,296 possible Huffman codes in a 32-bit implementation. This in turn results in a huge binary tree being produced for compression and decompression, which results in a significant slowdown whenever this needs to be traversed, sorted or modified in any way.

This is supported by the times taken to compress and decompress the written language file, lorem ipsum. The results show that when there are significantly less characters overall, such as in written language, where less than 100 different characters can be expected, the 32-bit implementation performs significantly better than against randomly generated files. The 32-bit implementation remains slower than the 8-bit implementation, but is many orders of magnitude faster than its equivalent random file. This is very likely also affected by the repetition of patterns found in written language, as this will allow for fewer unique entries to be created during compression for the 32-bit implementation.

The key to improving the 32-bit implementation would be to create a much more efficient method of constructing a binary tree. While a more efficient construction method would benefit both the 8-bit and 32-bit methods, the 32-bit method would benefit exponentially more as files got larger. It is possible that these methods could have similar performance, given an efficient enough tree construction method.

6.1.2 Compression Ratio Comparison

Input File	Ratio/Dictionary Size (Base / 8-bit)	Ratio/Dictionary Size (Improvement / 32-bit)
Lorem Ipsum (100KB)	0.53 / 1KB	0.34 / 70KB
Random (100KB)	1 / 3KB	0.46 / 540KB
Random (200KB)	1 / 3KB	0.49 / 1,080KB
Random (300KB)	1 / 3KB	0.51 / 1,621KB

For the comparison of compression ratios, the compression ratio (output file size divided by input file size) between in the input file and the output Huffman code file was measured separately from the size of the dictionary output alongside it. This was done to help illustrate the advantages and disadvantages of the 32-bit implementation compared to the 8-bit implementation, as it brings to the forefront the key issue: the dictionary.

Following on from the speed comparison, the key issue here can be traced back to the Huffman tree becoming too large and too many potential codes being created. The results in the table above show that, while the compression ratio on the file is better when using the 32-bit implementation, this is entirely offset by the huge dictionary required to decode it.

There is, however, potential for the 32-bit implementation to find use in an effective manner. The largest potential dictionary that could be created for a 32-bit Huffman encoding would be gigabytes in size, so if given a file of significant enough size, in the area of terabytes, the size of the dictionary would become less significant and the overall compression ratio may be more effective than the standard 8-bit implementation. With the time it would take such a large file to compress and decompress, as outlined in the speed comparison, it possible that the 8-bit implementation would be preferable even in this scenario.

The data obtained also shows that compression was more effective across the board on written language. This is most likely for similar reasons to the increase in speed found on written word. Less total potential characters combined with more repeating patterns means that it is more likely a code can be reused, thus increasing its frequency and reducing its size, resulting in a smaller file overall.

6.2 Analysis of LZ Implementation

The metrics analysed for the LZ Implementation were speed and memory usage. In order to achieve this, several files were generated to be compressed. The speed of these files was measured using the same timing circuit as for the Huffman implementation, and the memory usage was measured using the windows tool “Performance Monitor” (Microsoft, 2013).

6.2.1 Speed Comparison

Input File	Time Taken (Base) (s)	Time Taken (Improvement) (s)
File 1 (2KB)	5.783	8.327
File 2 (4KB)	11.211	16.986
File 3 (8KB)	23.803	36.943

As can be seen from the data above, the base version, which utilised a dictionary re-initialisation method, outperformed the pruning-based improvement implementation in speed. While both methods seemed to increase linearly with the size of the input file, the pruning-based implementation consistently took approximately 50% longer than the base implementation.

This is mostly likely due to two key factors in the processing of the dictionary: frequency of modification and cost of modification.

The first of these factors, frequency of modification, gives an advantage to the base modification in that it does not need to modify and reset the dictionary nearly as often as the improvement. Due to the fact that the base implementation resets the dictionary to its base 256 entries, it will be able to perform multiple insertions before it reaches the dictionary maximum limit and another reset is required. This in turn means that less total time is spent resetting the dictionary, as the dictionary will not need to be reset nearly as much. The process for resetting the dictionary to its base is also fast, as no searching or complex operations are required, simply 256 insert operations.

The second of these factors, cost of modification, refers to the resource cost of handling the dictionary when it reaches its maximum size. The cost for resetting the dictionary to its default 256 entries is also rather low. While it may seem that performing all of the insert operations required in order to fill up the dictionary once it has been reset may seem large and costly, this method results in a lower cost of modification than the pruning-based implementation. This is due to the fact that in order to create a code or compress a byte, the dictionary must be searched for the relevant entry. This is unavoidable and one of the costliest operations that must be performed, which in turn means that inserting at this stage, by comparison, takes very little time. This, combined with the additional operations required by the improvement to track frequency and generate weight, means that overall, there is very little difference in the cost of modification between the base version and the improvement.

This factor is increased due to the improvements need to search the dictionary and calculate the weight for each entry before any can be pruned. This means that for each loop after the dictionary reaches its max and begins to be pruned, an additional search must be executed every time. This adds up to a large chunk of time being spent searching the dictionary that the base implementation does not need to do.

These two factors result in the base version performing operations of similar cost to the improvement, but significantly less often, resulting in a faster overall speed.

6.2.2 Memory Comparison

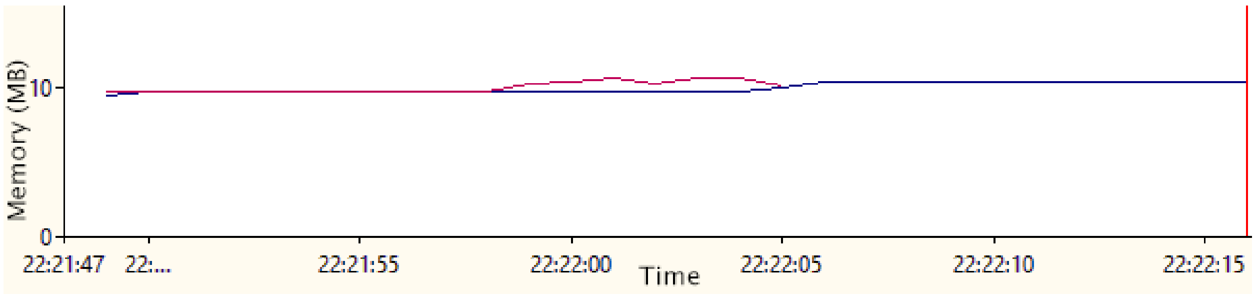

Fig 2. Memory Usage Comparison of Base Implementation (Red) and Improvement (Blue)

Fig show the graph of time and memory usage (MB) for the duration of the compression, where the base implementation is shown in red and the improvement is shown in blue. As the files used in compression were relatively small, the increases and decreases in memory usage were also small. The key factors to take from this are the maximum memory usage, and the pattern of the memory usage: how it increased and decreased over time.

The difference in memory usage was only slight between the two implementations, however there were some slight key differences between them.

The first of these was the pattern of memory usage. As can be seen in fig 2, the red base implementation’s memory usage would increase until it hit its maximum entries, whereupon it would then dramatically drop down as the dictionary was cleared. This resulted in periods of high memory usage, followed by periods of low memory usage. The improvement implementation, shown in blue, however, maintained a much steadier memory usage for its duration, due to only single dictionary entries being modified at any one time. Because of its changes in memory usage, the base implementation has resulted in a lower average memory usage across its duration than the improvement implementation.

The second of these was the maximum memory used. The maximum memory used during compression was slightly lower in the improvement implementation than in the base implementation. This was due to the improvement’s prioritisation of removing larger entries, ensuring that as much memory was conserved on each prune as possible. While the improvement does have a lower maximum memory usage, it is not particularly significant, with both implementations using very similar amounts of memory.

Considering these two differences, there does not seem to be much of a significant difference between the two implementations. If the dictionary were to get sufficiently large, it is possible the difference in maximum memory would increase with it, however this does not seem viable when combined with the slower speed of the improvement.

7 Conclusion

7.1 Discussion of Results

Even though the outcomes of the tests for both proposed methods were negative, I still believe the project to have been a success overall. Much has been learned from the results gathered on the advantages and disadvantages. While the negatives outweigh the positives for both improvements implemented, positives can still be identified from the results, with the compression ratios potentially improving with file size in the Huffman Coding implementation, and the reduction in maximum memory usage on the LZ implementation. As they are now, neither of the improvements would be used over the base version, but the potential for where they could be taken is there.

All the aims for the project were completed successfully. Various compression and encoding methods were researched and analysed to gain an understanding of their use-cases, advantages and disadvantages. This information was then used to identify and implement two improvements onto existing methods. Tests were then carried out against these new improvements in order to ascertain the potential use-cases, advantages and disadvantages of them.

I believe the results obtained from this project are meaningful, and while there are certainly areas that could be improved upon as well as issues that could be resolved, such as the LZ decompression issue that had to be left due to results taking precedent with concern to time management. All results obtained show a clear pattern and give insight into why these implementations would or would not be used.

7.2 Potential Applications

While the improvements implemented do not have any major obvious potential applications, there are still some areas in which they may see use.

For the Huffman Coding implementation, the key issues lay in the computing time in building and reading the binary tree, as well as the overall size of the output dictionary. As such, it may be effective in use where the input file is significantly large to the point that the dictionary size does not adversely affect the output size. It may also be effective in a large file with a small number of different combinations, in which the binary tree would be relatively small. In this scenario, the advantage of the larger codes would be received without the time investment in building the tree.

The LZ dictionary improvement could potentially see use in a system with extremely limited memory. In this scenario, the lower maximum memory usage could potentially allow for larger files to be compressed than if the base version were to be used.

7.3 Possible Extensions

A possible extension of the Huffman Coding implementation would be to develop a variable length code assignment implementation. As was seen in the results obtained, larger code sizes will eventually become more efficient with larger files, so it may be possible to implement a version of Huffman Coding that would encode the file with whichever code size was most effective for that file.

A key extension would be to focus on improving the efficiency in operations for each of the implementations, in particular the LZ implementation. It is possible that the loss in speed could be largely mitigated by use of a different, less costly technique to find and prune the weighted entries. If a method was implemented that could prune entries without the need for a costly search operation, it is entirely possible that this new method could be comparable to the current base method.

Another extension would be to create an implementation of the Huffman Coding improvement using the method of writing the whole binary tree to a file for decompression, as opposed to the dictionary writing method used in this project. It would be valuable to see the difference in dictionary size between these two methods, and whether using this binary tree writing method would help to mitigate the issue of the dictionary size.

8 Critical Reflection

8.1 Issues Relevant to Project Success

When this project began, it was initially very vague in scope and goal. While the aim was always to create an improvement of a data compression or encoding technique, a specific could not be decided upon without research into potential areas with room for a viable improvement to be implemented. This resulted in the need for an initial project specification that did not explicitly state the goals or intended outcome of the project. As such, an updated project specification was made once the goals had been decided upon.

There was always a slight concern with regards to 'competing' against the biggest players in compression, as it is an industry that has been around for a very long time as far as computing history goes. There is a huge amount of ongoing, long-standing research and developments in the compression industry, which made finding an area that could potentially be improved upon difficult, since so much thought had been put into this area over so much time. This meant there was always a risk to the success of the project, as finding a valid improvement was always going to be a challenge. In order to overcome this, I worked closely with my project supervisor and took onboard advice at various stages.

Because of all this prior work and history in this area, there was always a concern of the new approaches implemented not resulting in improvements. It was decided to go ahead with the project despite this concern, as even if the results came out negative, the results would still be worth gathering in order to understand why this may be the case, and potentially where the key areas that could be improved may be. On reflection, I could have attempted a safer project, however the fact that this line of inquiry had some risk to it has pushed me to explore novel solutions and helped to develop my creativity.

It was also due to this industry competition that it was decided to create two implementations of each proposed approach – a base and an improvement. This added a level of complexity to the project, but was deemed necessary in order to gather fair results. It would not be feasible to create an improvement based on industry source code, as it would not be realistic for my code to be of the same efficiency and polish of the industry standards. This meant that implementing both versions was necessary in order to level the playing field.

The recent COVID-19 pandemic has raised some significant challenges in the completion of this project. The disruption caused by having to move home, combined with the lack of access to the university library, adversely affected my working schedule. As the library does not have all texts in digital form, I was also unable to access some of the research materials I would normally be able to. I overcame this issue by defining a more rigorous development schedule and ensuring that more time was dedicated to researching the right areas. I also spent time ensuring I had a good working environment, which greatly aided me in the long term.

The use of SCRUM was extremely useful during this project, as it allowed me to adapt to incoming additional work and keep the project running efficiently. The ability to change sprints and modify particular sprint requirements based on priorities at the time was vital in completing the project in an effective manner. This became particularly vital with the above-mentioned COVID-19 pandemic, which required a major shifting of work and prioritisation, something that would have been significantly harder using a more rigid framework such as Waterfall.

Given opportunity to repeat this project, I would ensure that a more thorough testing regime was implemented, in order to ensure that the more obscure issues and errors were caught as soon as possible, such as the issue in the LZ Dictionary decompression stage that unfortunately went unnoticed for too long. I feel like this is the most important change I would make in the development process, and I have gained a greater knowledge of the importance of thorough verification due to this.

8.2 Issues Relevant to Professional and Ethical Concerns and Aspects of Personal Development Planning

A lot of knowledge was gained in various areas over the duration of this project. This has helped me to gain a firm grasp of the areas of computing my interests lie in, and has aided me in researching areas that I may not have otherwise done.

This project has given me respect for the process of improvement in software engineering, particularly over the large time frames found in compression. It is important for works to build upon one another, taking the novel or effective parts of a previous work and improving on it to create a new work. I have also learnt the value of performing research, even if the results may not be positive. As long as the ideas are reasonable and worth pursuing, they will be worth publishing regardless, despite the temptation to only publish positive results.

During the research process, a huge amount was learned about many compression and encoding techniques, as well as much of the mathematic theory that sits behind all of them. This has given me a much greater insight into how files are stored and transferred in computer systems, as well as an insight to some of the most base-level data storage mechanisms. This knowledge will be very useful in the area of computing I previously worked in, and aim to go into after graduating: systems administration.

A huge amount of knowledge and experience was gained in planning and project management. As this is by far the largest single project I have undertaken, I have learned a significant amount about techniques to aid in long-term development, such as rapid prototyping and time-management techniques such as SCRUM. These techniques are ubiquitous in the software industry, and as such will be developed and improved upon in any software role after graduating.

References

Blelloch, G. (2013). *Introduction to Data Compression.* Retrieved from https://www.cs.cmu.edu/~guyb/realworld/compression.pdf

Burrows, M., & Wheeler, D. (1994). *A block sorting lossless data compression algorithm.* Retrieved from https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf

Cleary, J., & Witten, I. (1984). Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, 396 - 402.

Fenwick, P. (2007). Burrows–Wheeler compression: Principles and reflections. *Theoretical Computer Science*, 200 - 219.

GitHub. (n.d.). *GitHub: About*. Retrieved from GitHub: https://github.com/about

Huffman, D. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 1098 - 1101.

Lempel, A., & Ziv, J. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 530 - 536.

Microsoft. (2013, 02 22). *Windows Performance Monitor*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc749249(v=ws.11)

Microsoft. (2018, 04 17). *Description of the "Lorem ipsum dolor sit amet" text that appears in Word Help*. Retrieved from Microsoft Support: https://support.microsoft.com/en-us/help/114222/description-of-the-lorem-ipsum-dolor-sit-amet-text-that-appears-in-wor

Microsoft. (n.d.). *Visual Studio*. Retrieved from Microsoft: https://visualstudio.microsoft.com/vs/

Prechelt, L. (1999). Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences. *Communications of the ACM*, 109 - 112.

Pu, I. M. (2005). *Fundamental data compression.* Elsevier Science & Technology.

Ryabko, B. Y. (1980). Data compression by means of a "book stack”. *Problems of Information Transmission*, 265 - 269.

Scrum. (n.d.). *What Is Scrum?* Retrieved from Scrum: https://www.scrum.org/resources/what-is-scrum

Standard C++ Foundation. (n.d.). *Big Picture Issues*. Retrieved from ISO C++: https://isocpp.org/wiki/faq/big-picture#zero-overhead-principle

Storer, J., & Szymanski, T. (1982). Data compression via textual substitution. *Journal of the ACM*, 928–951.

The Git Project. (n.d.). *Git: About*. Retrieved from Git: https://git-scm.com/about

Vanderjack, B. (2015). *The Agile edge : managing projects effectively using Agile Scrum.* New York: Business Expert Press.

Welch, T. (1984). A Technique for High-Performance Data Compression. *Computer*, 8 - 19.

Yuan, J. (2011). The combinational application of LZSS and LZW algorithms for compression based on Huffman. *Proceedings of 2011 International Conference on Electronics and Optoelectronics* (pp. 397 - 399). Dalian: IEEE.

Zeeh, C. (2003). *The Lempel Ziv Algorithm.* Retrieved from http://mat.hjg.com.ar/tic/img/LempelZivReport.pdf

Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 337-343.

APPENDIX 1A: Project Specification (original)

PROJECT SPECIFICATION - Project (Technical Computing) 2019/20

Student:	**Joe Mitchell Jones**

Date:	**21/10/19**
Supervisor:	**Simon Andrews**
Degree Course:	**Software Engineering BEng**
Title of Project:	**Investigating data compression optimisation techniques and methods**

Elaboration

Data compression is an extremely important area of computing; being able to more efficiently store and move data will always be very valuable in a system. Finding the ideal method for compressing data will always differ on what it is that is being compressed and the purpose of the compression. The aim of this project will be to identify what features of a given compression provide good performance for its particular job, then to create an implementation to improve and optimise on these compression methods. The type of data to be compressed will depend on the initial investigation into compression algorithm implementations and software. All data used in the project will either be in the public domain or dummy data.

I aim to create a program that will be able to compress and decompress data, outputting the new compressed or decompressed data to the user in the form of a file. Details about the compression will then be displayed, such as time spent compressing and compression ratio achieved. The project will aim to be cross platform and work on a majority of devices.

Main details gathered will include: The speed at which data is compressed, the reduction in size achieved and complexity of the methods used. These metrics will be compared between my implementation and current software options, experimenting to see the differences in all the above on various data-sets. These results would then be analysed to determine the success of my implementation.

Project Aims

- Investigate current methods of compression
- Investigate implementations of these methods and their use cases
- Make use of planning and SCM tools such as GitHub to organise the project
- Analyse current methods to determine the features that provide good performance
- Create an implementation of a compression method based on the analysis done
- Design and run a set of experiments to compare the performance of the new implementation with current software options.
- Record changes made to algorithm implementations and the effects they make

Project deliverable(s)

The work will involve one or more of the following possibilities:

- Modifying and improving an existing compression algorithm implementation
- Developing a new implementation of a compression algorithm

The exact nature of the work will depend on the outcome of the initial investigation into compression algorithms and software.
This will result in a deliverable that is an implementation of a compression algorithm written in C++. Users will be able to input data to be either compressed or decompressed and receive their output along with details concerning the compression process i.e. time and reduction in size.

Action plan

Task	**Milestone Date**
Investigate Feasibility	18/10/19
Investigate SCM and planning tools	25/10/19
Research first method	01/11/19
Submit information review	06/12/19
Implement first method	15/11/19
Research second method	22/11/19
Implement second method	06/12/19
Research third method	13/12/19
Implement third method	03/01/20
Create optimised compression algorithm implementation 1. Implement first version 2. Optimise and improve on algorithm implementation	16/02/20 (Overall) 1. 24/01/20 2. 15/02/20
Submit provisional contents page	21/02/20
Design and run experiments	28/02/20
Submit critical evaluation draft	27/03/20
Sections of draft report	27/03/20
Turnitin submission	22/04/20
Physical and electronic submission	23/04/20
Demonstration	Before 12/05/20

BCS Code of Conduct

I confirm that I have successfully completed the BCS code of conduct on-line test with a mark of 70% or above. This is a condition of completing the Project (Technical Computing) module.

Signature: [illegible]

Publication of Work

I confirm that I understand the "Guidance on Publication Procedures" as described on the Bb site for the module.

Signature: [illegible]

GDPR

I confirm that I will use the "Participant Information Sheet" as a basis for any survey, questionnaire or participant testing materials. This form is available on the Bb site for the module.

Signature: [illegible]

Ethics

Complete the SHUREC 7 (research ethics checklist for students) form below. If you think that your project may include ethical issues that need resolving (working with vulnerable people, testing procedures etc.) then discuss this with your supervisor as soon as possible and comment further here.

Both you and your supervisor need to sign the completed SHUREC 7 form.

Please contact the project co-ordinator if further advice is needed.

RESEARCH ETHICS CHECKLIST FOR STUDENTS (SHUREC 7)

This form is designed to help students and their supervisors to complete an ethical scrutiny of proposed research. The SHU Research Ethics Policy should be consulted before completing the form.

Answering the questions below will help you decide whether your proposed research requires ethical review by a Designated Research Ethics Working Group.

The final responsibility for ensuring that ethical research practices are followed rests with the supervisor for student research.

Note that students and staff are responsible for making suitable arrangements for keeping data secure and, if relevant, for keeping the identity of participants anonymous. They are also responsible for following SHU guidelines about data encryption and research data management.

The form also enables the University and Faculty to keep a record confirming that research conducted has been subjected to ethical scrutiny.

For student projects, the form may be completed by the student and the supervisor and/or module leader (as applicable). In all cases, it should be counter-signed by the supervisor and/or module leader, and kept as a record showing that ethical scrutiny has occurred. Students should retain a copy for inclusion in their research projects, and staff should keep a copy in the student file.

Please note if it may be necessary to conduct a health and safety risk assessment for the proposed research. Further information can be obtained from the Faculty Safety Co-ordinator.

General Details

Name of student	Joe Mitchell Jones
SHU email address	b5025654@my.shu.ac.uk
Course or qualification (student)	BEng Software Engineering
Name of supervisor	Simon Andrews
email address	s.andrews@shu.ac.uk
Title of proposed research	Investigating data compression optimisation techniques and methods
Proposed start date	26/09/19
Proposed end date	23/04/20

Brief outline of research to include, rationale & aims (250-500 words).	Data compression is an extremely important area of computing; being able to more efficiently store and move data will always be very valuable in a system. Finding the ideal method for compressing data will always differ on what it is that is being compressed and the purpose of the compression. The aim of this project will be to identify what features of a given compression provide good performance for its particular job, then to create an implementation to improve and optimise on these compression methods. The type of data to be compressed will depend on the initial investigation into compression algorithm implementations and software. All data used in the project will either be in the public domain or dummy data. I aim to create a program that will be able to compress and decompress data, outputting the new compressed or decompressed data to the user in the form of a file. Details about the compression will then be displayed, such as time spent compressing and compression ratio achieved. The project will aim to be cross platform and work on a majority of devices. Main details gathered will include: The speed at which data is compressed, the reduction in size achieved and complexity of the methods used. These metrics will be compared between my implementation and current software options, experimenting to see the differences in all the above on various data-sets. These results would then be analysed to determine the success of my implementation. • Investigate current methods of compression • Investigate implementations of these methods and their use cases • Make use of planning and SCM tools such as GitHub to organise the project • Analyse current methods to determine the features that provide good performance • Create an implementation of a compression method based on the analysis done • Design and run a set of experiments to compare the performance of the new implementation with current software options. Record changes made to algorithm implementations and the effects they make
Where data is collected from individuals, outline the nature of data, details of anonymisation, storage and disposal procedures if required (250-500 words).	There will be no collection of data from any individuals. Any research for this project will be gathered from academic sources such as books or papers. Any data used for testing will be public domain or dummy data.

1. Health Related Research Involving the NHS or Social Care / Community Care or the Criminal Justice Service or with research participants unable to provide informed consent

Question	**Yes/No**

1. Does the research involve? • Patients recruited because of their past or present use of the NHS or Social Care • Relatives/carers of patients recruited because of their past or present use of the NHS or Social Care • Access to data, organs or other bodily material of past or present NHS patients • Foetal material and IVF involving NHS patients • The recently dead in NHS premises • Prisoners or others within the criminal justice system recruited for health-related research***** • Police, court officials, prisoners or others within the criminal justice system***** • Participants who are unable to provide informed consent due to their incapacity even if the project is not health related	No
2. Is this a research project as opposed to service evaluation or audit? *For NHS definitions please see the following website* http://www.hra.nhs.uk/documents/2013/09/defining-research.pdf	No

If you have answered **YES** to questions **1 & 2** then you **must** seek the appropriate external approvals from the NHS, Social Care or the National Offender Management Service (NOMS) under their independent Research Governance schemes. Further information is provided below.
NHS https://www.myresearchproject.org.uk/Signin.aspx

***** All prison projects also need National Offender Management Service (NOMS) Approval and Governor's Approval and may need Ministry of Justice approval. Further guidance at: http://www.hra.nhs.uk/research-community/applying-for-approvals/national-offendermanagement-service-noms/

NB FRECs provide Independent Scientific Review for NHS or SC research and initial scrutiny for ethics applications as required for university sponsorship of the research. Applicants can use the NHS proforma and submit this initially to their FREC.

2. Research with Human Participants

Question	**Yes/No**
Does the research involve human participants? This includes surveys, questionnaires, observing behaviour etc.	No
Question	**Yes/No**
1. *Note If YES, then please answer questions 2 to 10* *If NO, please go to Section 3*	
2. Will any of the participants be vulnerable? *Note: Vulnerable' people include children and young people, people with learning disabilities, people who may be limited by age or sickness, etc. See definition on website*	

3. Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants or will the study involve invasive, intrusive or potentially harmful procedures of any kind?	
4. Will tissue samples (including blood) be obtained from participants?	
5. Is pain or more than mild discomfort likely to result from the study?	
6. Will the study involve prolonged or repetitive testing?	
7. Is there any reasonable and foreseeable risk of physical or emotional harm to any of the participants? *Note: Harm may be caused by distressing or intrusive interview questions, uncomfortable procedures involving the participant, invasion of privacy, topics relating to highly personal information, topics relating to illegal activity, etc.*	
8. Will anyone be taking part without giving their informed consent?	
9. Is it covert research? *Note: 'Covert research' refers to research that is conducted without the knowledge of participants.*	
10. Will the research output allow identification of any individual who has not given their express consent to be identified?	

If you answered **YES only** to question **1,** the checklist should be saved and any course procedures for submission followed. If you have answered **YES** to any of the other questions you are **required** to submit a SHUREC8A (or 8B) to the FREC. If you answered **YES** to question **8** and participants cannot provide informed consent due to their incapacity you must obtain the appropriate approvals from the NHS research governance system. Your supervisor will advise.

3. Research in Organisations

Question	**Yes/No**
1. Will the research involve working with/within an organisation (e.g. school, business, charity, museum, government department, international agency, etc.)?	No
2. If you answered YES to question 1, do you have granted access to conduct the research? *If YES, students please show evidence to your supervisor. PI should retain safely.*	
3. If you answered NO to question 2, is it because: A. you have not yet asked B. you have asked and not yet received an answer C. you have asked and been refused access. *Note: You will only be able to start the research when you have been granted access.*	

4. Research with Products and Artefacts

Question	**Yes/No**
1. Will the research involve working with copyrighted documents, films, broadcasts, photographs, artworks, designs, products, programmes, databases, networks, processes, existing datasets or secure data?	Yes

2. If you answered YES to question 1, are the materials you intend to use in the public domain? *Notes: 'In the public domain' does not mean the same thing as 'publicly accessible'.* • *Information which is 'in the public domain' is no longer protected by copyright (i.e. copyright has either expired or been waived) and can be used without permission.* • *Information which is 'publicly accessible' (e.g. TV broadcasts, websites, artworks, newspapers) is available for anyone to consult/view. It is still protected by copyright even if there is no copyright notice. In UK law, copyright protection is automatic and does not require a copyright statement, although it is always good practice to provide one. It is necessary to check the terms and conditions of use to find out exactly how the material may be reused etc.* *If you answered YES to question 1, be aware that you may need to consider other ethics codes. For example, when conducting Internet research, consult the code of the Association of Internet Researchers; for educational research, consult the Code of Ethics of the British Educational Research Association.*	Yes
3. If you answered NO to question 2, do you have explicit permission to use these materials as data? *If YES, please show evidence to your supervisor.*	
4. If you answered NO to question 3, is it because: A. you have not yet asked permission B. you have asked and not yet received and answer C. you have asked and been refused access. *Note: You will only be able to start the research when you have been granted permission*	A/**B/C**

Adherence to SHU policy and procedures

Personal statement	
I can confirm that: – I have read the Sheffield Hallam University Research Ethics Policy and Procedures – I agree to abide by its principles.	
Student	
Name: Joe Mitchell Jones	Date: 25/10/19
Signature: [illegible]	
Supervisor or other person giving ethical sign-off	
I can confirm that completion of this form has not identified the need for ethical approval by the FREC or an NHS, Social Care or other external REC. The research will not commence until any approvals required under Sections 3 & 4 have been received.	
Name: Simon Andrews	Date: 25/10/19
Signature: [illegible]	

APPENDIX 1B: Project Specification (post moderation)

PROJECT SPECIFICATION - Project (Technical Computing) 2019/20

Student:	**Joe Mitchell Jones**
Date:	**25/10/19**
Supervisor:	**Simon Andrews**
Degree Course:	**Software Engineering BEng**
Title of Project:	**Investigating data compression optimisation techniques and methods**

Elaboration

Data compression is an extremely important area of computing; being able to more efficiently store and move data will always be very valuable in a system. Finding the ideal method for compressing data will always differ on what it is that is being compressed and the purpose of the compression. The aim of this project will be to identify what features of a given compression or encoding method provide good performance for its particular job, then to create an implementation to improve and optimise on these compression or encoding methods.
The type of data to be compressed will be text and binary data and all compression and encoding done will be lossless. All data used in the project will either be in the public domain or dummy data.
I aim to create a program that will be able to compress and decompress data, outputting the new compressed or decompressed data to the user in the form of a file. Details about the compression will then be displayed, such as time spent compressing and compression ratio achieved.
The project will aim to be cross platform and work on a majority of devices. Main details gathered will include: The speed at which data is compressed, the reduction in size achieved and complexity of the methods used. These metrics will be compared between my implementation and current software options, experimenting to see the differences in all the above on various data-sets. These results would then be analysed to determine the success of my implementation.

Project Aims

•Investigate current methods of compression
•Investigate implementations of these methods and their use cases
•Make use of planning and SCM tools such as GitHub to organise the project
•Analyse current methods to determine the features that provide good performance
•Create two implementations of a compression and encoding method based on the analysis done, one base and one improvement.
•Design and run a set of experiments to compare the performance of the implemented improvement to the base implementation.
•Record changes made to algorithm implementations and the effects they make

Project deliverable(s)

The work will consist of two implementations of an encoding method, as well as two implementations of a compression method. These will be one base level implementation of an existing compression or encoding method with no improvement and one improvement implementation with the proposed modifications to the method added. This means that there will be four total programs created.
This will result in a deliverable that consists of implementations of a compression method and an encoding method, written in C++. Users will be able to input data to be either compressed or

decompressed and receive their output along with details concerning the compression process i.e. time and reduction in size.

Action plan

Task	**Milestone Date**
Investigate Feasibility	18/10/19
Investigate SCM and planning tools	25/10/19
Research first method	11/11/19
Implement first method	26/11/19
Research second method	10/12/19
Implement second method	02/01/20
Submit information review	06/12/19
Implement first improvement	25/01/20
Implement second improvement	17/02/20
Submit provisional contents page	21/02/20
Design and run experiments	28/02/20
Submit critical evaluation draft	27/03/20
Sections of draft report	27/03/20
Turnitin submission	22/04/20
Physical and electronic submission	23/04/20
Demonstration	Before 12/05/20

BCS Code of Conduct

I confirm that I have successfully completed the BCS code of conduct on-line test with a mark of 70% or above. This is a condition of completing the Project (Technical Computing) module.

Signature: [illegible]

Publication of Work

I confirm that I understand the "Guidance on Publication Procedures" as described on the Bb site for the module.

Signature: [illegible]

GDPR

I confirm that I will use the "Participant Information Sheet" as a basis for any survey, questionnaire or participant testing materials. This form is available on the Bb site for the module.
Signature:

[illegible]

Ethics

Complete the SHUREC 7 (research ethics checklist for students) form below. If you think that your project may include ethical issues that need resolving (working with vulnerable people, testing procedures etc.) then discuss this with your supervisor as soon as possible and comment further here.
Both you and your supervisor need to sign the completed SHUREC 7 form.
Please contact the project co-ordinator if further advice is needed.

RESEARCH ETHICS CHECKLIST FOR STUDENTS (SHUREC 7)

This form is designed to help students and their supervisors to complete an ethical scrutiny of proposed research. The SHU Research Ethics Policy should be consulted before completing the form.

Answering the questions below will help you decide whether your proposed research requires ethical review by a Designated Research Ethics Working Group.

The final responsibility for ensuring that ethical research practices are followed rests with the supervisor for student research.

Note that students and staff are responsible for making suitable arrangements for keeping data secure and, if relevant, for keeping the identity of participants anonymous. They are also responsible for following SHU guidelines about data encryption and research data management.

The form also enables the University and Faculty to keep a record confirming that research conducted has been subjected to ethical scrutiny.

For student projects, the form may be completed by the student and the supervisor and/or module leader (as applicable). In all cases, it should be counter-signed by the supervisor and/or module leader, and kept as a record showing that ethical scrutiny has occurred. Students should retain a copy for inclusion in their research projects, and staff should keep a copy in the student file.

Please note if it may be necessary to conduct a health and safety risk assessment for the proposed research. Further information can be obtained from the Faculty Safety Co-ordinator.

General Details

Name of student	Joe Mitchell Jones
SHU email address	b5025654@my.shu.ac.uk
Course or qualification (student)	BEng Software Engineering
Name of supervisor	Simon Andrews
email address	s.andrews@shu.ac.uk

Title of proposed research	Investigating data compression optimization techniques and methods
Proposed start date	26/09/19
Proposed end date	23/04/20
Brief outline of research to include, rationale & aims (250-500 words).	Data compression is an extremely important area of computing; being able to more efficiently store and move data will always be very valuable in a system. Finding the ideal method for compressing data will always differ on what it is that is being compressed and the purpose of the compression. The aim of this project will be to identify what features of a given compression or encoding method provide good performance for its particular job, then to create an implementation to improve and optimise on these compression methods. The type of data to be compressed will be text and binary data and all compression and encoding done will be lossless. All data used in the project will either be in the public domain or dummy data. I aim to create a program that will be able to compress and decompress data, outputting the new compressed or decompressed data to the user in the form of a file. Details about the compression will then be displayed, such as time spent compressing and compression ratio achieved. The project will aim to be cross platform and work on a majority of devices. Main details gathered will include: The speed at which data is compressed, the reduction in size achieved and complexity of the methods used. These metrics will be compared between my implementation and current software options, experimenting to see the differences in all the above on various data-sets. These results would then be analysed to determine the success of my implementation. •Investigate current methods of compression •Investigate implementations of these methods and their use cases •Make use of planning and SCM tools such as GitHub to organise the project •Analyse current methods to determine the features that provide good performance •Create two implementations of a compression and encoding method based on the analysis done, one base and one improvement. •Design and run a set of experiments to compare the performance of the implemented improvement to the base implementation. •Record changes made to algorithm implementations and the effects they make
Where data is collected from individuals, outline the nature of	There will be no collection of data from individuals. Any research for this project will be gathered from academic sources such as books or

data, details of anonymisation, storage and disposal procedures if required (250-500 words).	papers. Any data used for testing will be public domain or dummy data.

1. Health Related Research Involving the NHS or Social Care / Community Care or the

Criminal Justice Service or with research participants unable to provide informed consent

Question	**Yes/No**
1. Does the research involve? • Patients recruited because of their past or present use of the NHS or Social Care • Relatives/carers of patients recruited because of their past or present use of the NHS or Social Care • Access to data, organs or other bodily material of past or present NHS patients • Foetal material and IVF involving NHS patients • The recently dead in NHS premises • Prisoners or others within the criminal justice system recruited for health-related research***** • Police, court officials, prisoners or others within the criminal justice system***** • Participants who are unable to provide informed consent due to their	No
2. Is this a research project as opposed to service evaluation or audit? *For NHS definitions please see the following website* http://www.hra.nhs.uk/documents/2013/09/defining-research.pdf	No

If you have answered **YES** to questions **1 & 2** then you **must** seek the appropriate external approvals from the NHS, Social Care or the National Offender Management Service (NOMS) under their independent Research Governance schemes. Further information is provided below.

NHS https://www.myresearchproject.org.uk/Signin.aspx

***** All prison projects also need National Offender Management Service (NOMS) Approval and Governor's Approval and may need Ministry of Justice approval. Further guidance at:

http://www.hra.nhs.uk/research-community/applying-for-approvals/national-offender-management-service-noms/

NB FRECs provide Independent Scientific Review for NHS or SC research and initial scrutiny for ethics applications as required for university sponsorship of the research. Applicants can use the NHS pro-forma and submit this initially to their FREC.

2. Research with Human Participants

Question	**Yes/No**
Does the research involve human participants? This includes surveys, questionnaires, observing behaviour etc.	No
Question	**Yes/No**
1. *Note If YES, then please answer questions 2 to 10* *If NO, please go to Section 3*	
2. Will any of the participants be vulnerable? *Note: Vulnerable' people include children and young people, people with learning disabilities, people who may be limited by age or sickness, etc. See definition on website*	
3. Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants or will the study involve invasive, intrusive or potentially harmful procedures of any kind?	
4. Will tissue samples (including blood) be obtained from participants?	
5. Is pain or more than mild discomfort likely to result from the study?	
6. Will the study involve prolonged or repetitive testing?	
7. Is there any reasonable and foreseeable risk of physical or emotional harm to any of the participants? *Note: Harm may be caused by distressing or intrusive interview questions, uncomfortable procedures involving the participant, invasion of privacy, topics relating to highly personal information, topics relating to illegal activity, etc.*	
8. Will anyone be taking part without giving their informed consent?	
9. Is it covert research? *Note: 'Covert research' refers to research that is conducted without the knowledge of participants.*	
10. Will the research output allow identification of any individual who has not given their express consent to be identified?	

If you answered **YES only** to question **1,** the checklist should be saved and any course procedures for submission followed. If you have answered **YES** to any of the other questions you are **required** to submit a SHUREC8A (or 8B) to the FREC. If you answered **YES** to question **8** and participants cannot provide informed consent due to their incapacity you must obtain the appropriate approvals from the NHS research governance system. Your supervisor will advise.

3. Research in Organisations

Question	Yes/No
1. Will the research involve working with/within an organisation (e.g. school, business, charity, museum, government department, international agency, etc.)?	No
2. If you answered YES to question 1, do you have granted access to conduct the research? *If YES, students please show evidence to your supervisor. PI should retain safely.*	
3. If you answered NO to question 2, is it because: A. you have not yet asked B. you have asked and not yet received an answer C. you have asked and been refused access. *Note: You will only be able to start the research when you have been granted access.*	

4. Research with Products and Artefacts

Question	Yes/No
1. Will the research involve working with copyrighted documents, films, broadcasts, photographs, artworks, designs, products, programmes, databases, networks, processes, existing datasets or secure data?	Yes
2. If you answered YES to question 1, are the materials you intend to use in the public domain? *Notes: 'In the public domain' does not mean the same thing as 'publicly accessible'.* • *Information which is 'in the public domain' is no longer protected by copyright (i.e. copyright has either expired or been waived) and can be used without permission.* • *Information which is 'publicly accessible' (e.g. TV broadcasts, websites, artworks, newspapers) is available for anyone to consult/view. It is still protected by copyright even if there is no copyright notice. In UK law, copyright protection is automatic and does not require a copyright statement, although it is always good practice to provide one. It is necessary to check the terms and conditions of use to find out exactly how the material may be reused etc.* *If you answered YES to question 1, be aware that you may need to consider other ethics codes. For example, when conducting Internet research, consult the code of the Association of Internet Researchers; for educational research, consult the Code of Ethics of the British*	Yes

3. If you answered NO to question 2, do you have explicit permission to use these materials as data? *If YES, please show evidence to your supervisor.*	
4. If you answered NO to question 3, is it because: A. you have not yet asked permission B. you have asked and not yet received and answer C. you have asked and been refused access.	**A/B/C**

Adherence to SHU policy and procedures

Personal statement	
I can confirm that: – I have read the Sheffield Hallam University Research Ethics Policy and Procedures	
Student	
Name: Joe Mitchell Jones	Date: 25/10/19
Signature: [illegible]	
Supervisor or other person giving ethical sign-off	
I can confirm that completion of this form has not identified the need for ethical approval by the FREC or an NHS, Social Care or other external REC. The research will not commence until any approvals required under Sections 3 & 4 have been received.	
Name: Simon Andrews	Date: 25/10/19
Signature: [illegible]	