

# **DEVELOPMENT OF AN AUDIO COMPRESSION TOOL**



## **PROJECT REPORT**

Submitted in partial fulfillment of the Requirement  
for the award of the degree of the

**Bachelor of Computer Science  
and Engineering  
Of  
Bharathiyar University, Coimbatore.**

Submitted by

**Mahanathan.S.R 0027K0180**

**Ramasamy.S 0027K0196**

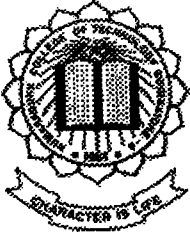
**Sureshkumar.K 0027K0205**

*Under the Guidance of*

**MR. M. NAGESHWARA GUPTHA B.E.,  
Lecturer, CSE Dept.**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
KUMARAGURU COLLEGE OF TECHNOLOGY  
COIMBATORE – 641 006.**

**MARCH 2004**



**KUMARAGURU COLLEGE OF TECHNOLOGY**  
**(Affiliated to Bharathiar University)**  
COIMBATORE – 641 006, TAMIL NADU, INDIA  
**Approved by AICTE, New Delhi - Accredited by NBA**



**Department of Computer Science and Engineering**

**CERTIFICATE**

This is to certify that the project entitled

**“DEVELOPMENT OF AN AUDIO COMPRESSION TOOL”**

has been submitted by

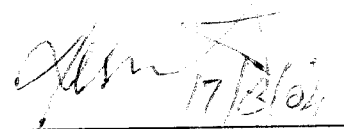
**MAHANATHAN S.R. (0027K0180)**

**RAMASAMY S. (0027K0196)**

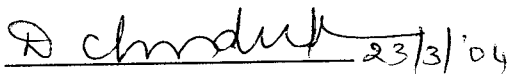
**SURESH KUMAR K. (0027K0205)**

in partial fulfillment of the requirements for the award of the degree of  
Bachelor of Engineering in Computer Science and Engineering of the  
Bharathiar University, Coimbatore – 641 046 during the academic year  
2003-2004

  
\_\_\_\_\_  
**Head of the Department**

  
\_\_\_\_\_  
**Project Guide**

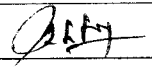
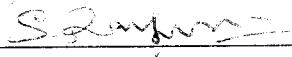

**Submitted for the university examination held on** 23/03/04

  
\_\_\_\_\_  
**Internal Examiner**

  
\_\_\_\_\_  
**External Examiner**

## DECLARATION

We, **Mahanathan.S.R 0027K0180, Ramasamy.S 0027K0196, Sureshkumar.K 0027K0205** here by declare that this project entitled "Development of an Audio Compression Tool" submitted to Kumaraguru College of Technology, Coimbatore(Affiliated to Bharathiar University) is a record of original work done by us under the supervision and guidance of Mr.M.Nageshwara Guptha B.E., Senior Lecturer, Department of Computer Science and Engineering.

NAME	REGISTRATION NUMBER	SIGNATURE
Mahanathan.S.R	0027K0180	
Ramasamy.S	0027K0196	
Sureshkumar.K	0027K0205	

  
17/3/04  
Countersigned:

Staff in charge: Mr.M.Nageshwara Guptha B.E,  
Lecturer,  
Department of Computer Science and Engineering,  
Kumaraguru College of Technology,  
Coimbatore.

Place: KCT, CBE - 06.

Date: 23/03/04 -

## *ACKNOWLEDGEMENT*

## **ACKNOWLEDGEMENT**

The exhilaration achieved upon the successful completion of any task should be definitely shared with the people behind the venture. This project is an amalgam of study and experience of many people without whose help this project would not have taken shape.

At the onset, we take this opportunity to thank the management of our college for having provided us excellent facilities to work with. We express our deep gratitude to our Principal Dr.K.K.Padmanabhan Ph.D., for ushering us in the path of triumph.

We are always thankful to our beloved Professor and the Head of the Department, Dr S.Thangasamy Ph.D., whose consistent support and enthusiastic involvement helped us a great deal.

We are greatly indebted to our beloved guide Mr.M.Nageshwara Guptha B.E., Lecturer, Department of Computer Science and Engineering for his excellent guidance and timely support during the course of this project. As a token of our esteem and gratitude, I honor him for his assistance towards this cause.

We also thank our project coordinator Ms.S.Chandrakala M.E., and our beloved class advisor Ms.Hema Guptha B.E., for their invaluable assistance.

We also feel elated in manifesting our deep sense of gratitude to all the staff and lab technicians in the Department of Computer Science and Engineering.

We feel proud to pay our respectful thanks to our Parents for their enthusiasm and encouragement and also we thank our friends who have associated themselves to bring out this project successfully.

*SYNOPSIS*

## **SYNOPSIS**

The project "Development of an Audio Compression Tool" aims to develop an audio compression tool. The significance of this product is that it allows the users to choose their own degree of compression. No compensation in quality of data is taken for the compression of the audio file. This product is highly suitable for those areas where the memory and transfer time are much more important. One such area is audio file with speech signals. This product highly suits for these types of inputs.

This product makes use of the redundancy available in the wave file format of audio signals. Further more, by a method of averaging and differencing the redundancy in the wave file is increased. This increased redundancy is used to compress the data. During compression, Run Length Encoding is used to compress the file with increased redundancy. Decompression involves the reverse process of averaging and differencing. Then the reverse process of Run Length Encoding is done to decompress the file to get the original file.

## *CONTENTS*



# CONTENTS

1) Introduction	1
1.1) Existing system and its limitations.	3
1.2) Proposed system and its advantages.	3
2) System Requirement Analysis	4
2.1) Product definition.	5
3) Software Requirement Specification.	6
3.1) Product Overview and Summary.	7
3.2) Development and Operating Environment.	7
3.3) External Interface.	7
3.4) Data Flow Diagrams.	8
3.5) Functional Specifications.	9
3.5.1) Compression.	9
3.5.2) Decompression.	10
3.6) Exception Handling.	11
3.7) Implementation Priorities.	11
4) Design Documentation.	12
4.1) Compression.	13
4.2) Decompression.	18
5) Product testing.	19
5.1) Unit Testing.	21
5.2) Integration Testing.	21
6) Future Enhancements.	22
7) Conclusion.	24
8) References.	26
9) Appendix.	28
9.1) Decomposition and Composition with Sample Values.	29
9.2) Sample Code.	34
9.3) Sample Output.	46

## *INTRODUCTION*

# 1. INTRODUCTION

Compression literally means the process of reducing the size of data. Compression avoids the need of large memory for storing the data. Apart from the less memory advantage, it also has advantage of less transfer time and speedy access. Less memory requirement and speedy transfer lead to many different compression standards. Compression has been categorized into two types; lossy and lossless.

Lossless compression means that the reproduced data is not changed in any way by the compression/decompression process; therefore, we do not have to worry about the quality of a lossless system – the output will be the same as the input. On the other hand, lossy compression systems by definition do make some change to the input – something is different. The trick is making that difference hard for the user to recognize. Lossy techniques may introduce any artifacts. None of these artifacts is easy to quantify.

## **1.1 EXISTING SYSTEM AND ITS LIMITATIONS:**

Both the lossy and lossless techniques are available nowadays. When audio compression is considered, the available techniques are meant for music files. The music files need a high clarity. Though lossy techniques are used, the deviations are hard to be recognized. So the reproduced data resembles almost the same as original data. They can be efficient but these compression standards do not allow the user to choose the level of compression. They try to compress the data as much as possible according to their design. But there are some places where the speedy transfer of data is much more significant when the quality of data is compared. At these instances the user is not allowed to choose the level or degree of compression.

When the user is allowed to choose the degree or level of compression, he can decide on the quality needed and the size needed. For instance, a wav file containing the data corresponding only to speech need not have that much quality as that of the music files. At these instances when user is allowed to choose his level of compression, it can be efficient.

## **1.2 PROPOSED SYSTEM AND ITS ADVANTAGES:**

This product compresses and decompresses the data according to user's decision. It allows the user to select the degree of compression according to his need. Three levels of compression: low, medium and high are presented for the user. According to the scenario, he has to choose a level. There will be deviation in reproduced data when these methods are used. Low level compression results in less reduction of size and high clarity but the high level compression reduces the data to a large extent with some loss of original data. So the high level compression is recommended only for the situations where the access speed or transfer time has a large significance than the quality of data.

# *SOFTWARE REQUIREMENT ANALYSIS*

## **2. SOFTWARE REQUIREMENT ANALYSIS**

System study is an activity that encompasses most of the tasks that we have collectively called computer system engineering. System study is conducted with the following objectives.

- Identify the needs.
- Evaluate the system concept for feasibility.
- Perform economic and technical analysis.
- Allocate function to hardware, software, people and other system elements.
- Create a system definition that forms the foundation for all subsequent engineering works.

### **PRODUCT DEFINITION:**

The main aim of this project is to develop a tool for compression and decompression that allows the user to decide on the degree of compression. The user selects a compression level from the menu presented. Accordingly the data gets compressed. The input for this tool should be a wav file. The wav file contains the digitized audio signals.

# *SOFTWARE REQUIREMENT SPECIFICATION*

### **3. SOFTWARE REQUIREMENT SPECIFICATION**

#### **3.1 PRODUCT OVERVIEW AND SUMMARY:**

This project aims at the development of a product to compress an audio file that allows the user to select his own Degree of compression on his data.

#### **3.2 DEVELOPMENT, OPERATING AND MAINTANENCE ENVIRONMENT:**

##### **Hardware Specification:**

Processor – Pentium III

RAM – 256 MB

Hard Disk – 10 GB

Floppy Drive – 1.44 MB

Monitor – 14" Monitor

##### **Software Specification:**

Operating System – Windows 98

Platform – Turbo C

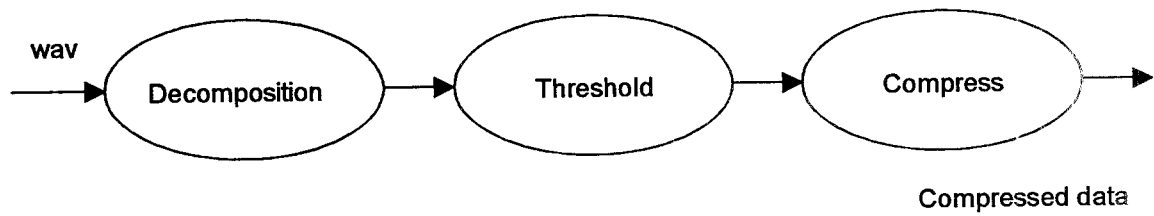
#### **3.3 EXTERNAL INTERFACE:**

The user is provided with a menu listing the compression levels: low, medium and high. Selection from the user is considered for the compression.

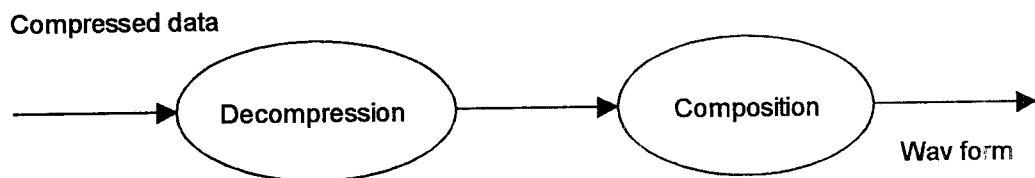


### 3.4 DATAFLOW DIAGRAM:

#### 3.4.1 COMPRESSION:



#### 3.4.2 DECOMPRESSION:



### **3.5 FUNCTIONAL SPECIFICATIONS:**

The project is divided into modules as:

#### **a) COMPRESSION:**

- 1) Reading a wav file.
- 2) Decomposition.
- 3) Thresholding.
- 4) Compression.

#### **b) DECOMPRESSION:**

- 1) Decompression.
- 2) Composition.

#### **3.5.1 COMPRESSION:**

Compression module comprises of the process of reading a wav file which is going to be the input to the product, decomposition of wav files, thresholding and compression.

##### **Reading a wav file:**

It comprises of the study on the wav files, the manner in which the data are stored under the format chunk and data chunk and the block structure of the stored data. It is the basic step in this project.

##### **Decomposition:**

The process of decomposition splits the input wav file into averages and differences. A recursive call to average and difference process is called till the whole data set is decomposed. At the end there can be a notable amount of redundancy which can be used in compression. But this process does not affect the quality of data.

### **Thresholding:**

Redundancy at the end of decomposition is not enough for the process of compression. So the process of thresholding is carried out on the decomposed data. Three levels of thresholding are presented for the user. He has to select a level. Accordingly the values are modified so that an increased redundancy is acquired in the decomposed data.

### **Compression:**

As a last step of compression, the thresholded data is compressed. To compress the data, Run length encoding is used. It makes use of the redundant data present adjacent in the data set.

### **3.5.2 DECOMPRESSION:**

The decompression module comprises of the process of decompressing the compressed data and composing the averages and differences into original values. Finally a wav format is presented as output with the format and data chunk included.

### **Decompression:**

The inverse process of Run length encoding is carried out to decompress the data. A set of averages and differences is produced at the end of this process.

### **Composition:**

Averages and differences are recursively acted upon to construct the original signal. Once this process is completed the implementation phase ends.

### 3.6 EXCEPTION HANDLING:

Some exceptions are expected to occur during the usage of this product such as: invalid file name and temporary space not available. If an invalid wav file is specified as input the compression and decompression process cannot be handled out. So an alert message is thrown out to indicate the error occurred. This product needs some MBs of free space for temporary use. If the temporary space is not available an alert message is thrown to indicate the status.

### 3.7 IMPLEMENTATION PRIORITIES:

While implementing, some priorities need to be followed as: Reading the wav file format should be done first. Because it gives the basic knowledge of how the data are stored and organized in a wav file. When composition and decomposition are considered, decomposition should be done prior to the composition. Decomposition implementation may affect the process of composition. And when compression and decompression are considered, compression should be done prior to decompression phase.

*DESIGN DOCUMENTATION*

### 3. DESIGN DOCUMENTATION

According to Webster, the process of design involves “conceiving and planning out in mind” and “making a drawing, pattern, or sketch of”. Generally, any work if done separately as modules will be easier to handle. This project involves the following modules:

1) COMPRESSION:

- a) Reading a wav file.
- b) Decomposition.
- c) Thresholding.
- d) Compression.

2) DECOMPRESSION:

- a) Decompression.
- b) Composition.

a) Reading a wav file:

The structure of a wav file holds an important part in this project. The analog signals are quantized and sampled at a suitable sampling frequency and are stored in a wav file. The wav files include two parts: format chunk and data chunk. These chunks present the details necessary to read or play a wav file. These chunks are analyzed to read the data from the wav file.

#### WAV FILE FORMAT

RIFF WAVE Chunk groupID = 'RIFF' riffType = 'WAVE'
Format Chunk ckID = 'fmt '
Data Chunk Ckid = 'data'

## Format chunk:

The structure of Format chunk is as follows:

```
Struct format_chunk
{
    Char ID[4];
    Long chunksize;
    Short formattag;
    Unsigned short channels;
    Unsigned long samplespersec;
    Unsigned long avgbytespersec;
    Unsigned short blockalign;
    Unsigned short Bitspersample;
};
```

The ID is always "fmt ". The chunksize field is the number of bytes in the chunk. This does not include the 8 bytes used by ID and Size fields. For the Format Chunk, chunksize may vary according to what "format" of WAVE file is specified (i.e., depends upon the value of Formattag).

WAVE data may be stored without compression. Different forms of compression may be used when storing the sound data in the Data chunk. With compression, each sample point may take a differing number of bytes to store. The Formattag indicates whether compression is used when storing the data.

If compression is used (i.e., Formattag is some value other than 1), then there will be additional fields appended to the Format chunk which give needed information for a program wishing to retrieve and decompress that stored data. The first such additional field will be an unsigned short that indicates how many more bytes have been appended (after this unsigned short). Furthermore, compressed formats must have a Fact chunk which contains an unsigned long indicating the size (in sample points) of the waveform after it has been

decompressed. If no compression is used (i.e., `FormatTag = 1`), then there are no further fields. The `Channels` field contains the number of audio channels for the sound. A value of 1 means monophonic sound, 2 means stereo, 4 means four channel sound, etc. Any number of audio channels may be represented. For multichannel sounds, single sample points from each channel are interleaved. A set of interleaved sample points is called a sample frame.

The actual waveform data is stored in another chunk. The `Samplespersec` field is the sample rate at which the sound is to be played back in sample frames per second (i.e., Hertz). The 3 standard MPC rates are 11025, 22050, and 44100 KHz, although other rates may be used.

The `Avgbytespersec` field indicates how many bytes play every second. `Avgbytespersec` may be used by an application to estimate what size RAM buffer is needed to properly playback the WAVE without latency problems. Its value should be equal to the following formula rounded up to the next whole number:

$$\text{Samplespersec} * \text{Blockalign}$$

The `Blockalign` field should be equal to the following formula, rounded to the next whole number:

$$\text{Channels} * (\text{Bitspersample} / 8)$$

Essentially, `Blockalign` is the size of a sample frame, in terms of bytes. (e.g., a sample frame for a 16-bit mono wave is 2 bytes. A sample frame for a 16-bit stereo wave is 4 bytes.

The `Bitspersample` field indicates the bit resolution of a sample point (i.e., a 16-bit waveform would have `Bitspersample = 16`). One, and only one, `Format Chunk` is required in every WAVE.



## Data chunk:

The format of data chunk is as follows:

```
Struct data_chunk
{
    Char ID [4];
    Long chunksize;
    Unsigned Char waveformat;
};
```

The ID always holds the value 'Data' in it. It represents that the following data corresponds the data part. Chunksize indicates the size of the data chunk; the total number of bytes in the data chunk.

### b) Decomposition:

Decomposition of a wav file includes the process of splitting it into averages and differences. This process is carried out in order to increase the redundancy of the data so that the compression technique can make use of this redundancy.

The decomposition is carried as follows:

- 1) Find the average of the adjacent data.
- 2) Find the difference between the average and the value.
- 3) Fill the first part with averages and the second part with differences.
- 4) Repeat the process till all values averaged and differenced.

### c) Thresholding:

After the completion of decomposition, the adjacent data may be slightly different from each other. When these values are replaced with some threshold values, we can have an increased redundancy. This increased redundancy increases the compression ratio. But this should be done carefully so that the quality of original value should not be affected. At this point, the user is provided with a menu listing the levels of compression. His choice of compression is considered for the compression.

### d) Compression:

Run-length encoding (RLE) is a very simple form of data compression encoding. It is based on simple principle of encoding data. This principle is to every stream which is formed of the same data values (repeating values is called a *run*) i.e. sequence of repeated data values is replaced with count number and a single value. This intuitive principle works best on certain data types in which sequences of repeated data values can be noticed; RLE is usually applied to the files that a contain large number of consecutive occurrences of the same byte pattern.

#### Principle of RLE:

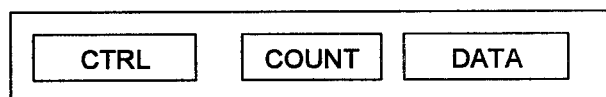
The basic RLE principle is that the run of characters is replaced with the number of the same characters and a single character.

Example:

before:                    R T A A A A S D E E E E E

after RLE compression: R T \*4A S D \*5E.

The format used in RLE technique is:



In the above example each stream of the same characters is being replaced with the number of characters, single character and character \*, at the first sight usage of \* may seem redundant but characters \* which represents control character (which may variously implemented) are necessary because some streams consist of the same characters or number of character is too small, so this special character represents that next characters represents the number of repetition and character after the number is the character which will be repeated. If control character or a run of control characters (in our example \*) are in a stream which is being encoded then one extra character is coded. It is important to realize that the encoding process is effective only if there are sequences of 4 or more repeating characters because three characters are used to conduct RLE so for instance coding two repeating characters would lead to expansion and coding three repeating characters wouldn't cause compression or expansion.

#### e) Decompression:

Constructing the original values from the compressed data is quite easier in the RLE technique. The input data are read and the following steps are carried out:

- 1) Read the data.
- 2) If not a control data, then store the data in the destination file.
- 3) If it is a control character, then the next value is stored as count.
  - a. The next value is stored count times in the destination file.
- 4) These steps are repeated till all data are processed.

#### f) Composition:

Once the values are decompressed, the original values should be composed from the decompressed values. Composing the values is just the reverse process of decomposition. The average and the difference are added and stored and the average and difference are subtracted and stored. This process results in the restoration of original values. This is a lossy compression technique but the loss is hard to be identified.

*TESTING*

## **5. TESTING**

Testing is an activity to verify that a correct system is being built and is performed with the intent of finding faults in the system. Testing is an activity, however not restricted to being performed after the development phase is complete. But this is to be carried out in parallel with all stages of system development, starting with requirement specification. Testing results once gathered and evaluated, provide a qualitative indication of software quality and reliability and serve as a basis for design modification if required.

System Testing is a process of checking whether the development system is working according to the original objectives and requirements. The system should be tested experimentally with test data so as to ensure that the system works according to the required specification. When the system is found working, test it with actual data and check performance.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. The increasing visibility of software as a system element and the attendant "cost" associated with a software failure are the motivation forces for a well planned, thorough testing.

## 5.1 UNIT TESTING:

Each and every program needed to be tested, which is known as unit testing. The module interface is to be tested to ensure that information properly flows in and out of the unit under test. The local data structure is to be examined to ensure that data condition stored temporarily maintains integrity throughout the unit's lifetime. Boundary condition is tested thoroughly. All independent paths through the control structure are to be exercised to ensure that all the statements in the modules have been executed at least once. All error handling paths are to be tested.

### Test case:

- 1) Testing of improper formats of files entered.
- 2) Testing for incorrect values for the fields, which are below or above the range.

## 5.2) INTEGRATION TESTING:

Once the modules are tested individually under the unit testing strategy, it is necessary to put all these modules together. It is here that the data can be lost across the interface. One module can have an inadvertent, adverse effect on another. Integration testing is a systematic technique for constructing the program structure while at the same time conducting test to uncover errors associated with interfacing. The objective is to take unit-testing modules and build a program structure that has been dictated by the design.

### Test case:

- 1) Wave files alone should be passed as an input to decompose and compose modules. They can very well work with non-wave files. So they are checked for their correctness.
- 2) Different sized files are checked for different degrees of compression across different modules and found to be valid.

*FUTURE SCOPE OF ENHANCEMENTS*

## **6. FUTURE SCOPE OF ENHANCEMENTS**

Level of compression and quality of data reproduced are always inversely proportional. This product provides three levels of compression for the user. Each level produces a different quality of reproduction. With high degree of compression, the data deviation will be higher with this method of averaging and differencing. So, in future the threshold levels provided along the user option can be enhanced so that the deviation is reduced. This results in a much more effective compression standard.

Also the current working environment of the project is 16-bit. In future this can be made to work in 32-bit environments with only slight modifications.



*CONCLUSION*

## **7. CONCLUSION**

With this product in use, we can have our own degree of compression according to our requirement. We can have the memory and transfer time considered, before going for a degree of compression. It will certainly be a successful product in this hi-tech world if the transfer time is considered significant along with the quality of data. The most significant part of this product, i.e. ,the degree of compression is chosen by the user, is tested and confirmed to be error free and effective.

The programming techniques used in the design of the system provide a scope for further expansion and implementation of any changes, which may occur in the future.

## *REFERENCES*

## 8. REFERENCES

- John F. Koegel Buford, "**Multimedia Systems**", Pearson Education Pvt. Ltd., 2002, Sixth Indian Reprint.
- Dennis M Ritchie, "**The C Programming Language**", Prentice Hall of India Private Limited, June 1999, Second Edition.
- [www.borg.com](http://www.borg.com) visited on 25<sup>th</sup> November, 2003.
- [www.neurotraces.com](http://www.neurotraces.com) visited on 25<sup>th</sup> November, 2003.
- [www.members.optushome.com](http://www.members.optushome.com) visited on 15<sup>th</sup> December,2003.

*APPENDIX*

## 9. APPENDIX

### 9.1 DECOMPOSITION AND COMPOSITION WITH SAMPLE VALUES:

#### 9.1.1 DECOMPOSITION:

The process of decomposition is done based on the averages and differences between the data. For example, consider the following eight data. First the averages are stored and differences are stored. This is repeated till all data are processed.

**Original data :**

3	5	4	8	13	7	5	3
---	---	---	---	----	---	---	---

**Step I:**

4	6	10	4	-1	-2	3	1
---	---	----	---	----	----	---	---

**Step II:**

5	7	-1	3	-1	-2	3	1
---	---	----	---	----	----	---	---

**Step III:**

6	-1	-1	3	-1	2	3	1
---	----	----	---	----	---	---	---

### 9.1.2 COMPOSITION:

To retrieve the original data from the compressed data, we have to follow the reverse process of decomposition.

**Decomposed data:**

6	-1	-1	3	-1	-2	3	1
---	----	----	---	----	----	---	---

**Step I:**

5	7	-1	3	-1	-2	3	1
---	---	----	---	----	----	---	---

**Step II:**

4	6	10	4	-1	-2	3	1
---	---	----	---	----	----	---	---

**Step III:**

3	5	4	8	13	7	5	3
---	---	---	---	----	---	---	---

**Original data:**

3	5	4	8	13	7	5	3
---	---	---	---	----	---	---	---

### 9.1.3 THRESHOLDING:

The process of thresholding is done to increase the redundancy so that they can be used in the compression process. It involves the process of replacing the data which are less than a selected level with the selected data. For example, consider the following data:

**Low:**

Threshold: **-2**

**Decomposed data:**

6    -1    -1    3    -1    -2    3    1

**Thresholded data:**

6    -1    -1    3    -1    -2    3    1

**Step I:**

5    7    -1    3    -1    -2    3    1

**Step II:**

4    6    10    4    -1    -2    3    1

**Step III:**

3    5    4    8    13    7    5    3

**Original data:**

3    5    4    8    13    7    5    3

This method involves a very slight deviation from the original data. But the redundancy produced is very much less.



**Medium:**

Threshold: -1

**Decomposed data:**

6    -1    -1    3    -1    **-2**    3    1

**Thresholded data:**

6    -1    -1    3    -1    -1    3    1

**Step I:**

5    7    -1    3    -1    -2    3    1

**Step II:**

4    6    10    4    -1    -2    3    1

**Step III:**

3    5    4    **7**    13    7    5    3

**Original data:**

3    5    4    **8**    13    7    5    3

This method involves a slight deviation from original data which is shown blocked.

**High:**

Threshold: 1

**Decomposed data:**

6   -1   -1   3   -1   -2   3   1

**Thresholded data:**

6   1   1   3   1   -2   3   1

**Step I:**

7   5   1   3   1   1   3   1

**Step II:**

8   6   8   2   1   1   3   1

**Step III:**

9   7   7   5   11   5   3   1

**Original data:**

3   5   4   8   13   7   5   3

This method has a very high deviation from the original data. But the redundancy produced is very much high which can be effectively used for compression.

## 9.2 SOURCE CODE

### HEADERFORMAT.H

```
#ifndef headerfmt
#define headerfmt
struct fmt
{
    char id[4];
    long csize;
    short fmntag;
    unsigned short channels;
    unsigned long samplespersec;
    unsigned long avgbytespersec;
    unsigned short blockalign;
    unsigned short bitspersample;
};
struct newfmt
{
    char id[4];
    long csize;
    short fmntag;
    unsigned short channels;
    unsigned long samplespersec;
    unsigned long avgbytespersec;
    unsigned short blockalign;
    unsigned short bitspersample;
    int level;
};
struct data
{
    char id[4];
    long csize;
    //unsigned char wavfmtdata[100];
};
void error()
{
    printf("Invalid Input\n");
    printf("Check input values\n");
    exit(0);
}
#else
#error Header Included Already
#endif
```

## DECOMPOSE.H

```
#include<stdio.h>
#include<math.h>
void decompose(FILE *fp1,FILE *fp2,long size,long count)
{
    int *bread1,*bread2,*agv,*diff;
    int i,j,k;
    while(size>1)
    {
        printf("\n\n***ASA: %f\n\n",size);
        num=size/2;
        c1=8;
        for(i=1;i<=size/2;i++)
        {
            fscanf(fp1,"%d",&byte1);
            fscanf(fp1,"%d",&byte2);
            average=(byte1+byte2)/2;
            diff=byte1-average;

            fprintf(fp2,"%d\t",average);
            fprintf(temp,"%d\t",diff);

            printf("\nAvg : %d",average);
        }
        rewind(temp);
        while(num-->0)
        {
            fscanf(temp,"%d",&byte);
            fprintf(fp2,"%d\t",byte);
            printf("\nDiff : %d",byte);
        }
        rewind(fp1);
        rewind(fp2);

        printf("\n\n");

        while(c1-->0)
        {
            fscanf(fp2,"%d",&byte);
            fprintf(fp1,"%d\t",byte);
            printf("%d\t",byte);
        }
        rewind(fp1);
        rewind(fp2);
    }
}
```

```

        rewind(temp);
        size/=2;
    }
    return;
}

```

## COMPOSE.H

```

#include<stdio.h>
#include<math.h>

void compose(FILE *fp1,FILE *fp2,long count1,long size)
{
    int *bread, val[30000];
    int finc=0,temp[30000];
    int fin[30000],temp1;
    int i,j,k,n;
    int count,tot;

    clrscr();

    for(i=0;i<count1;i++) //writing into asavalues.txt
    {
        //fread(bread,2,1,fp1);
        fscanf(fp1,"%d",&bread);
        val[i]=bread;
        //fprintf(fp2,"%d ",*bread);
    }
    size=count1/2;

    label:
    n=pow(2,count);
    for(i=0;i<n;i++)
    {
        temp[i]=val[i];
        tot++;
    }
    while(i!=count1)
        temp[j++]=val[i++];
    for(i=0;i<tot;i++)
    {
        fin[finc]=temp[i]+temp[i];
        finc++;
        fin[finc]=temp[i]-temp[i];
        finc++;
    }
}

```

```

    }
    for(i=0;i<finc;i++)
        val[i]=fin[i];

    count++;
    finc=0;
    tot=0;
    j=0;
    size=size/2;
    //if(count!=3)
    if(size>1)
        goto label;
    for(i=0;i<count1;i++)
        fprintf(fp2,"%d\t",val[i]);

    return;
}

```

RLENC.H

```

#include<stdio.h>
#include<conio.h>
void encode(FILE *f1,FILE *f2)
{
    int byte1,byte2,i=0,j,k=0,l;
    FILE *f1,*f2,*f3;
    int temp[1024];
    int temp1[1024];
    int handle1,handle2;
    int count=1;
    printf("length = %d\n",len);
    while(i-1 < len)
    {
        fread(temp,2,1024,f1);
        if(temp[i] != temp[i+1])
        {
            if(count!=0)
            {
                temp1[k++]='*';
                itoa(count,ct,10);
                printf("ct = %c \t",ct[0]);
                //temp1[k++]=count;
                //strcpy(temp1[k++],ct[0]);
                temp1[k++] = ct[0];
                temp1[k++] = temp[i];
                printf("count = %d \n",count);
            }
        }
    }
}

```

```

        count=1;
    }
    else if (count == 1)
    {
        temp1[k++]=temp[i];
    }
    else if (count == 2)
    {
        temp1[k++]=temp[i];
        temp1[k++]=temp[i];
    }
    else
    {
        temp1[k++]=temp[i];
    }
}
else
{
    count++;
}
i++;
fwrite(temp1,2,1024,f2);
}
/*handle1=open("sss1.txt",O_RDONLY,0);
//handle2=creat("destin.txt",S_IREAD|S_IWRITE);
if((handle1 == -1) && (handle2 == -1))
{
    printf("can't open\n");
    exit(0);
}
while((count=read(handle1,buf,10)) != 0)
{
    //printf("%s",*buf);
    write(handle2,buf,10);
}
close(handle1);
close(handle2);*/
getch();
}

```

## COMPRESS.C

```
#include<stdio.h>
#include<alloc.h>
#include<conio.h>
#include"headrfmt.h"
#include"decompos.h"
#include"rlenc.h"
#include"compos.h"

void main(int count,char *argv[])
{
    FILE *f1,*f2,*f3;
    struct fmt fmt1;
    struct data d1;
    struct newfmt nfmt1;
    char *gid,*rifftype,*ptr1,*ptr2=".wav",*ptr3;
    long *size;
    int count1,option;
    int *bread;

    clrscr();

    if(count < 2)
    {
        error();
        exit(0);
    }

    ptr1=strstr(argv[1],ptr2);

    if(ptr1==NULL)
    {
        error();
        exit(0);
    }

    f1=fopen(argv[1],"rb+");
    f2=fopen("rec.srm","wb+");

    if((f1==NULL) || (f2==NULL))
    {
        printf("Invalid File Name\n");
        exit(0);
    }

    fread(gid,4,1,f1);
```





```

count1=d1.csize/fmt1.blockalign;
printf("\n\tSAMPLE FRAMES : %d\n",count1);
printf("\t*****asa*****\n");

printf("\t\tSelect the Degree of Compression\n");
printf("\t\t-----\n");
printf("\t\tEnter : \n");
printf("\t\t1 For Low Degree\n");
printf("\t\t2 For Medium Degree\n");
printf("\t\t3 For High Degree\n");
scanf("%d",&option);
nfmt1.level=option;
fwrite(&nfmt1,sizeof(nfmt1),1,f2);
fwrite(&d1,sizeof(d1),1,f2);
switch(option)
{
case 1:
    while(!feof(f1))
    {
        decompose(f1,f2,*size,count1);
        encode(f1,f2);
        fread(bread,2,1,f1);
        fread(bread,2,1,f1);
        fwrite(bread,2,1,f2);
    }
    break;
case 2:
    while(!feof(f1))
    {
        decompose(f1,f2,*size,count1);
        encode(f1,f2);
        fread(bread,2,1,f1);
        fread(bread,2,1,f1);
        fread(bread,2,1,f1);
        fwrite(bread,2,1,f2);
    }
    break;
case 3:
    while(!feof(f1))
    {
        decompose(f1,f2,*size,count1);
        encode(f1,f2);
        fread(bread,2,1,f1);
        fread(bread,2,1,f1);
        fread(bread,2,1,f1);
        fread(bread,2,1,f1);
        fwrite(bread,2,1,f2);
    }
}

```

```

        }
        break;
default:
    printf("Invalid option\n");
}

printf("Compressed\n");

fclose(f1);
fclose(f2);
return;
getch();
}

```

## DECOM.C

```

#include<stdio.h>
#include<alloc.h>
#include<conio.h>
#include"headrfmt.h"
#include"decompos.h"
#include"rlenc.h"
#include"compos.h"

void main(int argc,char *argv[])
{
    FILE *f1,*f2,*f3;
    struct fmt fmt1;
    struct data d1;
    struct newfmt nfmt1;
    char *gid,*rifftype;
    char *ptr1,*ptr2,*ptr3=".srm";
    long *size;
    int count,option;
    int *bread;
    clrscr();
    if(argc<2)
    {
        printf("Insufficient arguments\n");
        exit(0);
    }

    ptr1=strstr(argv[1],ptr3);

    if(ptr1==NULL)

```



```

fmt1.blockalign=nfmt1.blockalign;
printf("\t\tBITSPERSAMPLE : %d\n",nfmt1.bitspersample);
fmt1.bitspersample=nfmt1.bitspersample;
option=nfmt1.level;
fwrite(&fmt1,sizeof(fmt1),1,f2);

fread(&d1,sizeof(d1),1,f1);
fwrite(&d1,sizeof(d1),1,f2);
d1.id[4]='\0';
printf("\t\t\tDATA CHUNK\n");
printf("\t\t\t-----\n");
printf("\t\t\tDATA ID : %s\n",d1.id);
printf("\t\t\tCHUNK SIZE : %d\n",d1.csize);

count=d1.csize/fmt1.blockalign;
printf("\nSAMPLE FRAMES : %d\n",count);
printf("*****asa*****\n");

switch(option)
{
case 1:
    while(!feof(f1))
    {
        decode(f1,f2);
        compose(f1,f2);
        fread(bread,2,1,f1);
        fwrite(bread,2,1,f2);
        fwrite(bread,2,1,f2);
    }
    break;
case 2:
    while(!feof(f1))
    {
        decode(f1,f2);
        compose(f1,f2);
        fread(bread,2,1,f1);
        fwrite(bread,2,1,f2);
        fwrite(bread,2,1,f2);
        fwrite(bread,2,1,f2);
    }
    break;
case 3:
    while(!feof(f1))
    {
        decode(f1,f2);
        compose(f1,f2);
        fread(bread,2,1,f1);

```

```
        fwrite(bread,2,1,f2);
        fwrite(bread,2,1,f2);
        fwrite(bread,2,1,f2);
        fwrite(bread,2,1,f2);
    }
    break;
default:
    printf("Invalid option\n");
}
printf("Decompressed\n");
fclose(f1);
fclose(f2);
getch();
}
```

## 9.3 OUTPUT

### COMPRESSION

```
D:\ASA\FINPRO\LAST>compress sample.wav
```

Select the Degree of Compression

Enter :

- 1 For Low Degree
- 2 For Medium Degree
- 3 For High Degree

Given file is compressed.

```
D:\ASA\FINPRO\LAST>
```

### DECOMPRESSION

```
D:\ASA\FINPRO\LAST>decom sample.srm
```

Given file is decompressed.

```
D:\ASA\FINPRO\LAST>
```