File Zipper Software

Table of Contents

Introduction

Our main motive was to recreate the popular application "WinRAR" which is heavily used by people on a day to day basis.

Team Members:

1. Mohammad Yehya Hayati, K213309
2. Sufyan Abdul Rasheed, K213206

Background

The main research done was on the various different data structures and how to use them in our program. The data structures used in this project were:

- Nodes
- Linked Lists
- Stacks
- MinHeap/Priority Queue
- Huffman Trees

Their uses and functionality is further discussed in the later sections. However a brief summary is given as follows:

Nodes were used for the implementation of Linked Lists and Stacks.

Linked Lists were used for storing an undetermined amount of characters/pixels.

Stacks were used as a buffer to move nodes from a Linked List to a MinHeap/Priority Queue.

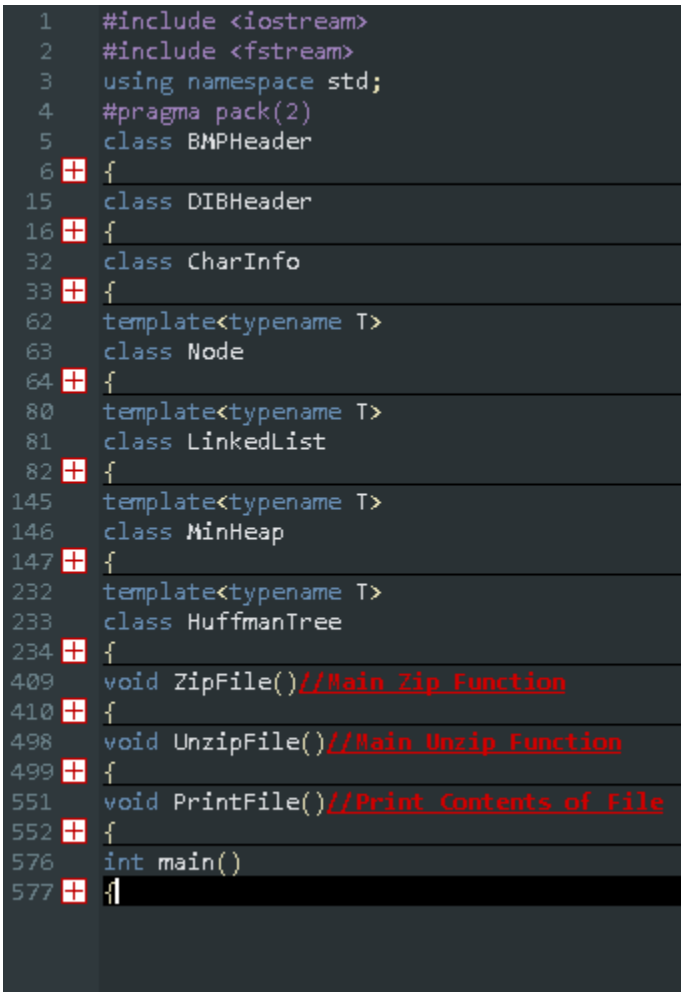
MinHeaps were used to store nodes with characters/pixels with a priority set on their frequency. This was also further used as a sorting class which was implemented by the functionality of HeapSort ().

Huffman Trees were used to generate a unique prefix code for every individual character/pixel, which was then used as a replacement for their ASCII codes.

The functionality of all the above mentioned points are combined into 2 functions; zipfile () and unzipfile ().

In addition to this, classes for the storing of data were created. For example, class CharInfo was created to store character information and class BMPHeader was created to store header files of images.

Here is an overview of all the different classes and data structures.



Implementation & Testing

(For the further inspection, see attached code)

First and foremost, we will look at the node class. This was needed as it acts as the baseline for all the data structures we will need to use. We made it a generic class to ensure usability of the same class for different file types. For us, we have supported 2 file types which are .txt files and 24-bit .bmp files. Here is our implementation of it.

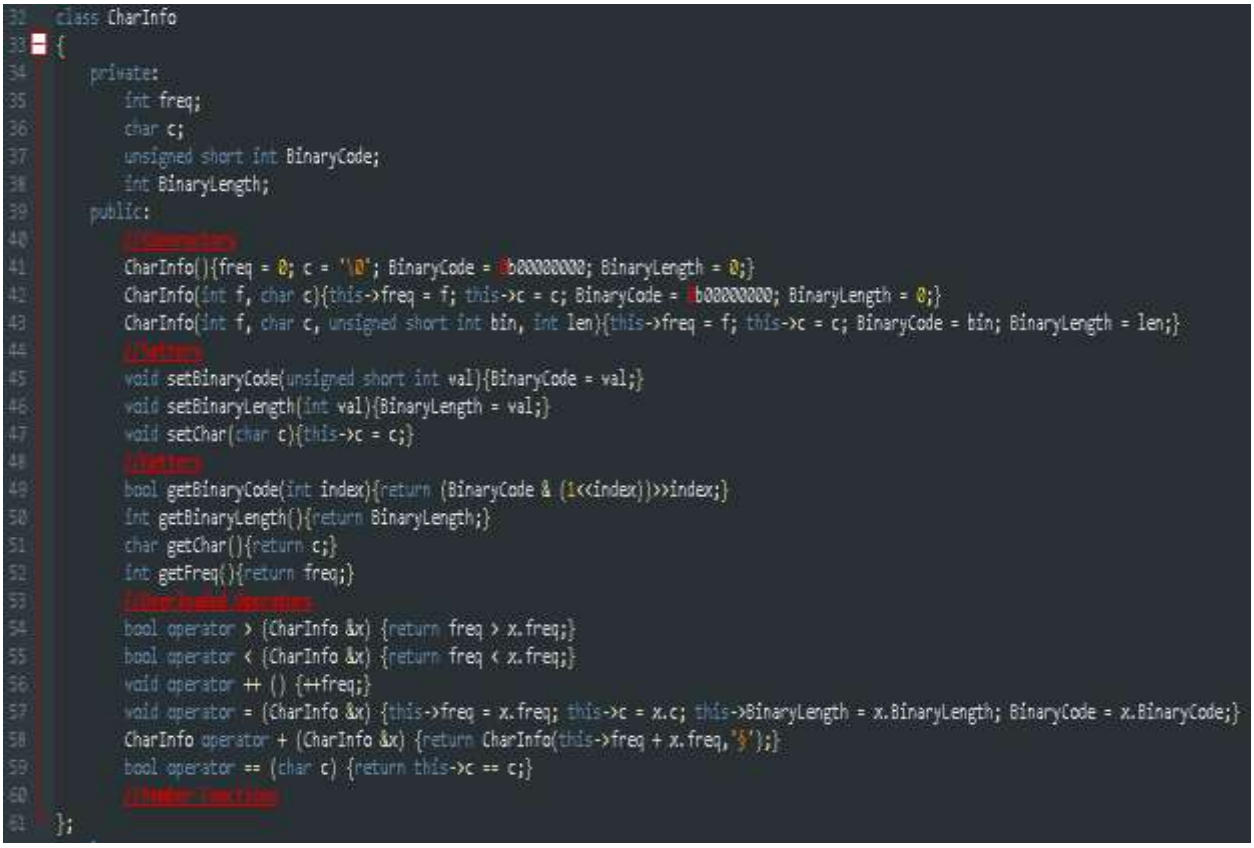
```
62  template<typename T>
63  class Node
64  {
65      public:
66          T data;
67          Node<T> *left, *right;
68      public:
69          //Constructors
70          Node(){left = right = NULL;}
71          Node(T data){left = right = NULL; this->data = data;}
72          //Destructors
73          ~Node(){left = right = NULL;}
74          //Overloaded Operators
75          bool operator > (Node<T> &x) {return data > x.data;}
76          bool operator < (Node<T> &x) {return data < x.data;}
77          void operator = (Node<T> *x) {*data = x->data;}
78          //Member Functions
79  };
```

Next we will take a look at the class that holds all the data. This is the CharInfo class. It contains four different and equally important details about any unique character. Its character, frequency, binary code, and the length of binary code. At the time of instantiation only the character and frequency are used and the binary code and length are in play later (during encoding).

However we faced a hurdle during this process, and solved it which resulted in the auxiliary space of our program to decrease from 16 bytes to 16 bits! More on this will be discussed later under the “Problems and Challenges” heading. Furthermore, using binary literals which were introduced in C++14, introduced elements of assembly code which we learnt in COAL and made accessing variables faster for our compiler.

Another thing to note is that this class can also be used for raster/bitmap images as they are also written in a similar format, which is 3 characters represent a pixel.

Here is our implementation of it.



```
class CharInfo
{
    private:
        int freq;
        char c;
        unsigned short int BinaryCode;
        int BinaryLength;
    public:
        //Constructors
        CharInfo(){freq = 0; c = '\0'; BinaryCode = 0b00000000; BinaryLength = 0;}
        CharInfo(int f, char c){this->freq = f; this->c = c; BinaryCode = 0b00000000; BinaryLength = 0;}
        CharInfo(int f, char c, unsigned short int bin, int len){this->freq = f; this->c = c; BinaryCode = bin; BinaryLength = len;}
        //Setters
        void setBinaryCode(unsigned short int val){BinaryCode = val;}
        void setBinaryLength(int val){BinaryLength = val;}
        void setChar(char c){this->c = c;}
        //Getters
        bool getBinaryCode(int index){return (BinaryCode & (1<<index))>>index;}
        int getBinaryLength(){return BinaryLength;}
        char getChar(){return c;}
        int getFreq(){return freq;}
        //Overloaded Operators
        bool operator > (CharInfo &x) {return freq > x.freq;}
        bool operator < (CharInfo &x) {return freq < x.freq;}
        void operator ++ () {++freq;}
        void operator = (CharInfo &x) {this->freq = x.freq; this->c = x.c; this->BinaryLength = x.BinaryLength; BinaryCode = x.BinaryCode;}
        CharInfo operator + (CharInfo &x) {return CharInfo(this->freq + x.freq,'$');}
        bool operator == (char c) {return this->c == c;}
        //Member Functions
};
```

Next we will discuss a bit on image files (specifically 24-bit bitmap files). Bitmap files contain a file header which stores important data of any image. For example, it contains the type of the image file or the offset of the image. It takes a total of 14 bytes of the starting of a bmp file. However, this is a problem in itself which will be discussed later in the “Problems and Challenges” heading.

Bitmap files contain another header that is called the DIB (Device Independent Bitmap) header contains the information of the image. For example, the width*height, bits per pixel, and even color depth. This is important since we need to know the width of an image to bypass a very famous problem, of compiler self-allocation, which will be discussed later in the “Problems and Challenges” heading.

Here is our implementation of it.

```
#pragma pack(2)
class BMPHeader
{
    private:
        char HeaderField[2];
        unsigned int size;
        unsigned int garbage;
        unsigned int imageOffset;
    public:
        unsigned int getImageOffset(){return imageOffset;}
};
class DIBHeader
{
    private:
        unsigned int hsize;
        int width;
        int height;
        unsigned short int colorplanes;
        unsigned short int bpp;
        unsigned int compression;
        unsigned int size;
        unsigned int hor_res;
        unsigned int ver_res;
        unsigned int colorDepth;
        unsigned int impColor;
    public:
        unsigned int getWidth(){return width;}
};
```

Then we needed a way to store all the characters from a file, but we don't exactly know how many characters there are. Therefore we needed to implement a Linked List class.

This Linked List class was made to fit the needs of a Linked List and as well as a Stack. Thus it has a particular set of functions as seen below.

```
80  template<typename T>
81  class LinkedList
82  {
83      Node<T> *head;
84      public:
85          //Constructors
86          LinkedList():head(NULL){}
87          LinkedList(T d){head = new Node<T>(d);}
88          //Destructor
89          ~LinkedList(){DeleteList();}
90          //Member Functions
91          void DeleteList()
92          {
...
104         void Insert(T d)
105         {
...
110         Node<T>* ReturnHead(){return head;}
111         void InsertChar(char ch)
112         {
...
126         Node<T>* Pop()
127         {
...
133         int SizeofList()
134         {
...
144 };
```

Then we needed a tree-like structure; one that can also semi-sort the nodes that we have created in the Linked List. Thus we used a MinHeap. For the sake of keeping the space complexity of our program at a minimum, we have used the shallow copy technique with removing the pointers from the Linked List to make sure there are no memory leaks.

We have also created a function that will be quite helpful when discussing the next data structure.

```
217     Node<T>* CreateHuffmanTree()
218     {
219         Node<T> *left, *right, *root;
220         while(top != 1)
221         {
222             left = ExtractRoot();
223             right = ExtractRoot();
224             root = new Node<T>(left->data+right->data);
225             root->left = left;
226             root->right = right;
227             Insert(root);
228         }
229         return ExtractRoot();
230     }
```

This small piece of code extracts the two smallest values of the MinHeap (using the logic of heap sort) and creates a new node that is the parent of

these two nodes, and then pushes the node back into the queue. This process repeats itself until there is one node left which will be the root node of our Huffman Tree.

Here is our implementation of a MinHeap.

```
145   template<typename T>
146   class MinHeap
147   {
148       Node<T> **arr;
149       int capacity;
150       int top;
151       int parent(int i) {return (i-1)/2;}
152       int left(int i) {return 2*i + 1;}
153       int right(int i) {return 2*i + 2;}
154       public:
155           //Constructors
156           MinHeap(){arr = NULL;}
157           MinHeap(int n)
158           {
163           //Destructor
164           ~MinHeap()
165           {
170           //Member Functions
171           int Capacity(){return capacity;}
172           void Insert(Node<T> *d)
173           {
185           void InsertList(LinkedList<T>& List) {for(int i = 0 ; i < capacity ; i++) Insert(List.Pop());}
186           Node<T>* ExtractRoot()
187           {
202           void MinHeapify(int i)
203           {
217           Node<T>* CreateHuffmanTree()
218           {
231   };
```

Next we will talk about the main class of our program. The Huffman tree is a very interesting tree in which the main data only exists at the leaves of the tree. This ensures that every relevant node has a unique code of traversal from the root of the tree. This can be used to our advantage as we can substitute these values as the new ASCII values of our characters.

To further explain, this is done by a simple process. Encode, Compress, Decode, and Decompress.

Encode basically generates a prefix binary code that is set to a leaf node along with the length of the code. This is done extremely quickly due to the use of bitwise operators working with binary literals as explained in the CharInfo section.

Compress is the function that reads a character and then stores its equivalent binary code into a 1 byte buffer. Then, once the buffer is full, it will write

the contents of the buffer into a new file which will be named the compressed file.

Decode is the function to read the header of a compressed file and recreate a Huffman tree to the T. This ensures that every character will be decoded with the exact same details as it was before the compression.

Decompression is the function to read the contents of a compressed file into a 1 byte buffer and traversing the tree until you reach a leaf node. The character in that leaf node is then written into a new file which will be named the decompressed file and should be equal to the original file.

Here is our implementation of it.

```
233  class HuffmanTree
234  {
235      Node<T>* root;
236      public:
237          //Constructors
238          HuffmanTree(){root = NULL;}
239          HuffmanTree(Node<T>* n){root = n;}
240          //Destructor
241          ~HuffmanTree(){DeleteHuffmanTree();}
242          //Helper Functions
243          DeleteHuffmanTree(){DeleteHuffmanTree(root);root=NULL;}
244          void DeleteHuffmanTree(Node<T>* r)
245          {
251          //Encoding & Compression Functions
252          void EncodeHuffmanCodes(fstream& file2 , LinkedList<T>& L1, bool NOWRITE = 0, int SKIP = 0)
253          {
262          void EncodeHuffmanCodes(Node<T> *r , unsigned short int &t , int index , fstream& File2 , LinkedList<T>& L, bool NOWRITE = 0)
263          {
289          void CompressFile(fstream& File1 , fstream& File2 , LinkedList<T>& L, int Width = 0, int SKIP = 0)
290          {
328          //Decoding and Decompression Functions
329          void DecodeHuffmanCodes(fstream& file2)
330          {
338          Node<T>* DecodeHuffmanCodes(fstream& file2, Node<T> *r, int size)
339          {
369          void DecompressFile(fstream& file2 , fstream& file3, int Width = 0, int SKIP = 0)
370          {
408  };
```

To sum up all that was said, a simple function named ZipFile can be created to execute these concepts.

Here is the implementation of it.

```
409  void ZipFile()//Main Zip Function
410  {
411      LinkedList<CharInfo> List;
412      fstream ReadFile, WriteFile;
413      string FileName = "";
414      char c;
415
416      cout << "Enter Name of the File to Zip: ";
417      cin >> FileName;
418
419      //Opening File To ZIP
420      ReadFile.open(FileName.c_str() , ios::in | ios::binary);
421      if(!ReadFile.is_open()) {cout << "No Such File Exists!";return;}
422
423      if(!FileName.compare(FileName.find('.'),4,".txt"))
424      {
449      else if(!FileName.compare(FileName.find('.'),4,".bmp"))
450      {
500      else cout << "Exception Thrown! Incorrect or Unsupported file type entered!";
501      ReadFile.close();
502  }
```

Similarly, to unzip a compressed file, we can create a new function named UnzipFile.

Here is the implementation of it.

```
503  void UnzipFile()//Main Unzip Function
504  {
505      fstream ReadFile, WriteFile;
506      string FileName = "";
507
508      cout << "Enter Name of the File to Unzip: ";
509      cin >> FileName;
510
511      //Opening Compressed File
512      ReadFile.open(FileName.c_str() , ios::in | ios::binary);
513      if(!ReadFile.is_open()) {cout << "No Such File Exists!";return;}
514
515      if(!FileName.compare(FileName.find('.'),4,".txt"))
516      {
530      else if(!FileName.compare(FileName.find('.'),4,".bmp"))
531      {
558      ReadFile.close();
559      cout << "File Successfully Unzipped with Name: " << FileName << "!\n";
560  }
```

Here is the main function.

```
int main()
{
    int choice;
    while(true)
    {
        cout << "\t\tFAST NUCES TXT File Zipper\n" << "1.Zip File\n2.Unzip File\n3.Print File\n4.Exit\nChoice: ";
        cin >> choice;
        if(choice == 1) ZipFile();
        else if (choice == 2) UnzipFile();
        else if (choice == 3) PrintFile();
        else if (choice == 4) break;
        cin.get();
        cin.get();
        system("CLS");
    }
    cout << "\nMade By Mohammad Yehya Hayati (K213309), Sufyan Abdul Rasheed (K213206)";
}
```

Here are some examples.

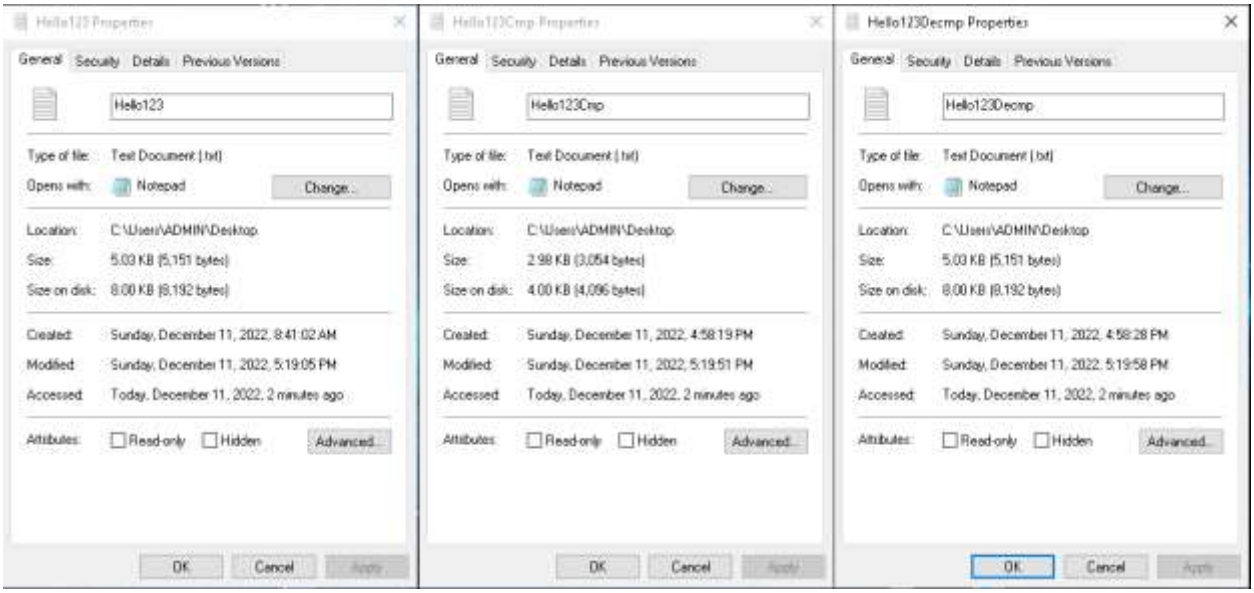
1. Original File:

Compressed File:

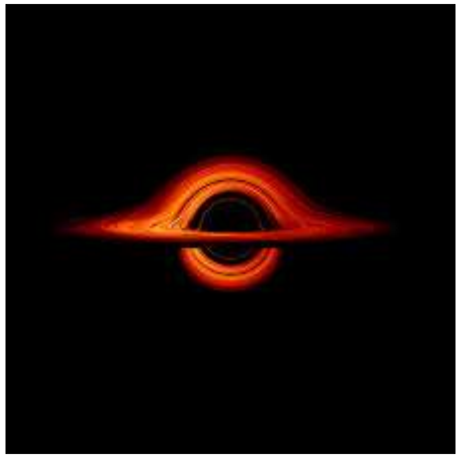
Decompressed File:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Cursus mattis molestie a iaculis at erat pellentesque adipiscing. Massa sapien faucibus et molestie ac feugiat sed. Convallis convallis tellus id interdum velit laoreet id. Adipiscing elit pellentesque habitant morbi tristique senectus et. Et malesuada fames ac turpis egestas sed tempus. Sed cras ornare arcu dui. Ac placerat vestibulum lectus mauris ultrices eros in cursus. Odio morbi quis commodo odio aenean sed. Nibh sit amet commodo nulla facilisi nullam vehicula ipsum. Blandit cursus risus at ultrices. Vel pretium lectus quam id leo in vitae turpis massa. Viverra ipsum nunc aliquet bibendum enim facilisis gravida neque convallis. Nullam non nisi est sit. Hac habitasse platea dictumst vestibulum. Risus pretium quam vulputate dignissim suspendisse in est ante. Nulla porttitor massa id neque aliquam. Purus gravida quis blandit turpis cursus in hac. Aliquam eleifend mi in nulla posuere sollicitudin aliquam ultrices sagittis. Sit amet nulla facilisi morbi tempus iaculis urna id volutpat. Volutpat ac tincidunt vitae semper. Quis imperdiet massa tincidunt nunc pulvinar sapien et ligula ullamcorper. Non tellus orci ac auctor augue mauris. Nisl scelerisque eu ultrices vitae. Sit amet nisl purus in mollis nunc sed id. Egestas sed tempus urna et pharetra. Auctor neque vitae tempus quam pellentesque nec nam aliquam. Augue eget arcu dictum varius duis at consectetur lorem donec. Eget nullam non nisi est sit amet facilisis magna. Odio ut enim blandit volutpat. Pretium viverra suspendisse potenti nullam ac tortor. At risus viverra adipiscing at. Bibendum est ultricies integer quis auctor. Eleifend donec pretium vulputate sapien. Cursus metus aliquam eleifend mi in nulla posuere. Eu tincidunt tortor aliquam nulla facilisi cras fermentum odio. Feugiat nisl pretium fusce id velit ut tortor pretium viverra. Posuere ac ut consequat semper viverra. Ipsum dolor sit amet consectetur adipiscing elit ut aliquam. Et tortor at risus viverra adipiscing at in tellus. Morbi leo urna molestie at. Non curabitur gravida arcu ac tortor dignissim convallis aenean. Eget velit aliquet sagittis id consectetur purus. Porttitor lacus luctus accumsan tortor posuere ac ut. Magna ac placerat vestibulum lectus mauris ultrices eros in cursus. Habitasse platea dictumst quisque sagittis purus. Turpis egestas sed tempus urna et pharetra pharetra massa. Donec adipiscing tristique risus nec feugiat in. Non curabitur gravida arcu ac. Quam nulla porttitor massa id neque aliquam. Urna nec tincidunt praesent semper feugiat. Sem viverra aliquet eget sit amet tellus cras adipiscing enim. Facilisis sed odio morbi quis commodo. Imperdiet nulla malesuada pellentesque elit eget gravida. Ipsum a arcu cursus vitae congue mauris rhoncus aenean vel. Et malesuada fames ac turpis egestas integer eget aliquet nibh. Amet justo donec enim diam vulputate ut. Faucibus pulvinar elementum integer enim neque. Sem et tortor consequat id porta nibh venenatis cras sed. Nulla pellentesque dignissim enim sit amet venenatis urna. Volutpat maecenas volutpat blandit aliquam etiam erat velit scelerisque. Sodales neque sodales ut etiam sit amet. Sagittis orci a scelerisque purus semper eget duis at. Amet tellus cras adipiscing enim eu. Odio euismod lacinia at quis risus sed. Nec dui nunc mattis enim ut tellus elementum sagittis. Lacus sed turpis tincidunt id aliquet. Vehicula ipsum a arcu cursus vitae congue mauris rhoncus aenean. Id ornare arcu odio ut sem nulla pharetra diam. Elit ut aliquam purus sit amet luctus venenatis. Ut enim blandit volutpat maecenas volutpat blandit aliquam etiam. Ullamcorper sit amet risus nullam eget. Cursus in hac habitasse platea dictumst quisque sagittis. Nullam vehicula ipsum a arcu. Aliquam sem et tortor consequat id porta nibh venenatis cras. Nisl rhoncus mattis rhoncus urna neque viverra justo nec ultrices. Amet purus gravida quis blandit turpis cursus. Sit amet justo donec enim diam vulputate ut. Facilisis gravida neque convallis a. Vel facilisis volutpat est velit egestas dui id. Habitasse platea dictumst quisque sagittis purus sit amet. Hac habitasse platea dictumst quisque. Est ullamcorper eget nulla facilisi etiam dignissim diam. Enim nulla aliquet porttitor lacus luctus accumsan tortor posuere. Quam id leo in vitae. Risus pretium quam vulputate dignissim suspendisse in. Dignissim suspendisse in est ante in nibh mauris cursus. Sit amet dictum sit amet justo donec enim. Turpis cursus in hac habitasse platea. Dui sapien eget mi proin sed libero enim. Varius vel pharetra vel turpis. Vitae tortor condimentum lacinia quis vel. Maecenas accumsan lacus vel facilisis volutpat est velit egestas dui. Egestas sed sed risus pretium quam vulputate dignissim suspendisse in. Tempus iaculis urna id volutpat lacus laoreet non curabitur gravida. At urna condimentum mattis pellentesque. At lectus urna duis convallis convallis tellus id. Mattis vulputate enim nulla aliquet porttitor lacus luctus accumsan tortor. Facilisis magna etiam tempor orci. Fringilla urna porttitor rhoncus dolor purus.

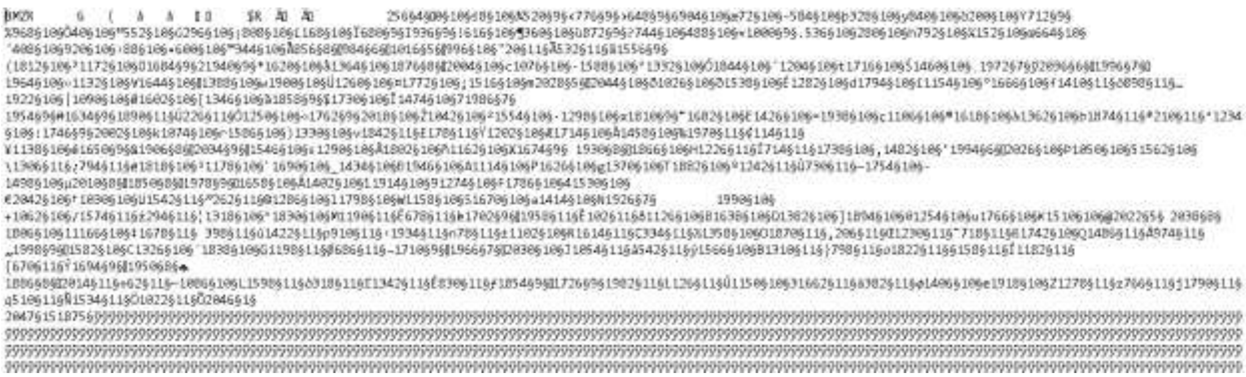
Comparison:



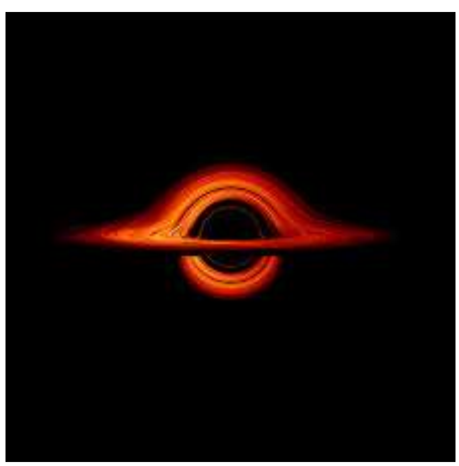
2. Original:



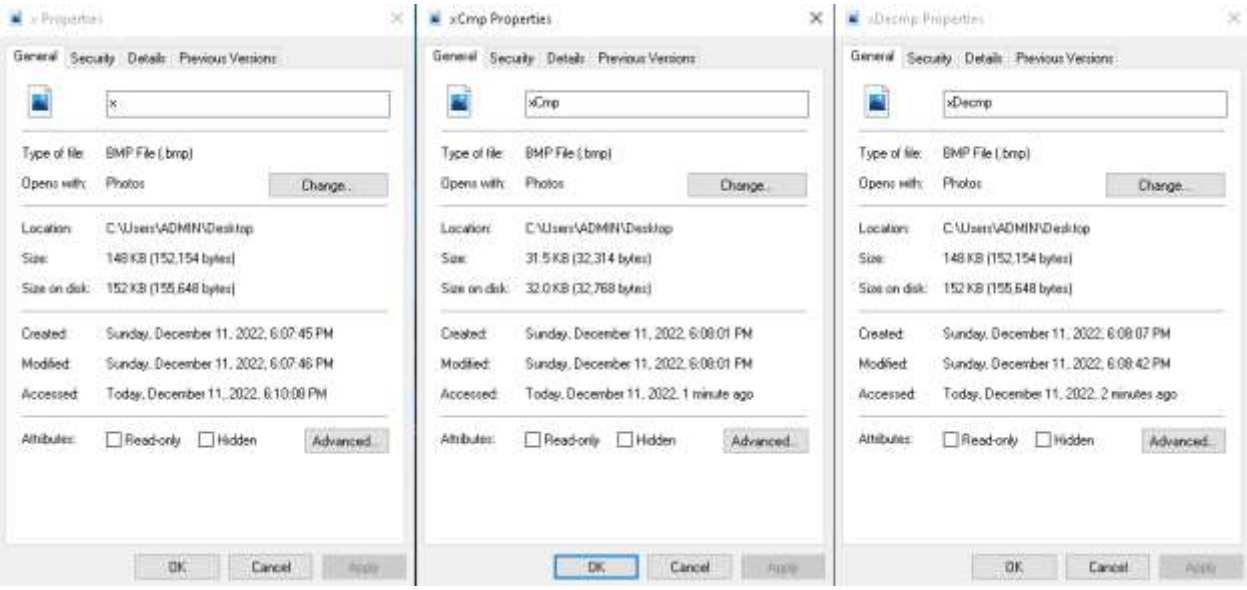
Compressed (In Text form):



Decompressed:



Comparison:



Problems and Challenges

There were many hurdles that we had to face to complete this project. The first being a filing error where the file pointer would go beyond the allowed boundaries. This was simply solved by opening the file using binary mode.

Another challenge was to reduce the auxiliary space used. This was done by replacing a Boolean array of size 16 to an unsigned short integer with size 2 using binary literals. This is one the rare cases where both time and space complexity decreases.

One major issue was that since the size of the BMP header was 14, the compiler self-allocates 2 bytes to make it a multiple of 4, but that would mess all the reading and would not compress tha file properly. There we had to use a preprocessor directive call pack (int). This would make it package all the bytes in multiple of the number specified.

Another great issue was that since the pixels are written in multiples of 3, and the width is not a multiple of 4, it would self-allocate enough bytes to make it a multiple of four. Therefore when reading or writing we had to make sure to deal with this issue using the concept of padding. We would calculate the amount of extra bytes beforehand and skip them when reading or write that many bytes of garbage when writing.

Conclusion and Breakdown

In the end, the outcome of the project is very satisfactory, and we have learnt a lot from participating this in this project.

Zipping File ----→Done by 17/Nov/2022

Unzipping File ----→Done by 22/Nov/2022

Fixed File out of Bounds----→Done by 22/Nov/2022

Fixed class self-allocation----→Done by 3/Dec/2022

Fixed file self-allocation----→Done by 4/Dec/2022

Made improvements----→Done by 9/Dec/2022

Thank You