Practical 1: Implementation of Shannon-Fano data compression algorithm.

Program to determine uniquely decodable codes.

```
#include<iostream>
#include<fstream>
#include <map>
#include<math.h>
using namespace std;
struct ele
{
	float fr;
	char ch;
	string code;
}data[100];

int isfound(string matchit,int n)
//for decoding by taking string and matching with table
{
	for(int i=0;i<n;i++)
	{
		if(matchit==data[i].code)	//if matched than return its index
			return i;
	}
	return -1;		//didnt found the matching code so return -1

}
string getbinary(int n)   //converting int to 8 bit binary
{
	string str;
	for (int i = 7; i >= 0; i--) {
	  int k = n >> i;
	  if (k & 1)
		str+="1";
```

```
        else
            str+="0";
    }
    return str;
}

void displaytable(int n)
{
       cout<<"\n\n Character\t"<<"Frequency\t"<<"Code\n";
       for(int i=0;i<n;i++)
       cout<<""<<data[i].ch<<"\t\t"<<data[i].fr<<"\t\t"<<data[i].code<<endl;
}

void process(int start,int end)  //partition with the help of arguments
{
       if(start == (end-1) )     //if only 2 elements left in the segments
       {
              data[start].code+="0";
              data[end].code+="1";
       }
       else if(start<end)
       {
              int fptr=start,bptr=end,mid;
              float fsum=data[start].fr,bsum=data[end].fr;
               //fsum(front elements sum) bsum(back elements sum)
              while(1)
              {
                     if(fsum>=bsum)      //if front is bigger than back
                            bsum+=data[--bptr].fr; //add another element to backsum
                     else if(fsum<bsum)  //if back is bigger than front
                            fsum+=data[++fptr].fr; /add another element to frontsum
                     if(fptr==bptr-1){  //found the optimal mid point for partition
                            mid=fptr;break;
                     }
```

```
            cout<<"\n Partition at index : "<<mid;
            for(int i=start;i<=mid;i++)      //concating 0 for left segment
                  data[i].code+="0";
            for(int i=mid+1;i<=end;i++)    //concating 1 for right segment
                  data[i].code+="1";
            process(start,mid);
            process(mid+1,end);
      }
}

int main()
{
      ifstream inFile;
      ofstream OutFile;
      std::map<char,int> trace; //dict type data structure
      char text;
      inFile.open("original.txt");
      if (!inFile)
      {
            cout << "Unable to open file";
            exit(1); // terminate with error
      }
      while (inFile >> text)    //read char and save to text
           trace[text]++;    //key=text  value=occurrence count

      int i=0;
      map<char, int>::iterator itr;
      cout << " KEY\tOCCURENCE\n";
      for (itr = trace.begin(); itr != trace.end(); ++itr)
      {
          cout <<" "<< itr->first<<"\t   " << itr->second << '\n';
          data[i].ch=itr->first; //assigning it to structure defined for process
          data[i++].fr=itr->second;
      }
```

```
inFile.close();
int n=i;
for(int i=0;i<n-1;i++)                  //sorting descending
{
       for (int j = 0; j < n-i-1; j++)
              if (data[j].fr < data[j+1].fr)
              {
                     ele temp;
                     temp=data[j];
                     data[j]=data[j+1];
                     data[j+1]=temp;
              }
}
process(0,n-1);
displaytable(n);
std::map<char,string> findcode;               //for mapping purpose
map<char,string>::iterator itr2;
for (int i=0;i<n;i++) {
   findcode[data[i].ch]=data[i].code;
//saving code in index as character so we can use later for retrieval
purpose
}

inFile.open("original.txt");                          //main input file
OutFile.open("Binarycoded.txt");
//original to binary(0 and 1) conversion according to codetable
cout<<"\nEncoded Binary Text : ";
while(inFile>>text)
{
       cout<<findcode[text];
       OutFile<<findcode[text];
}
inFile.close();
OutFile.close();
```

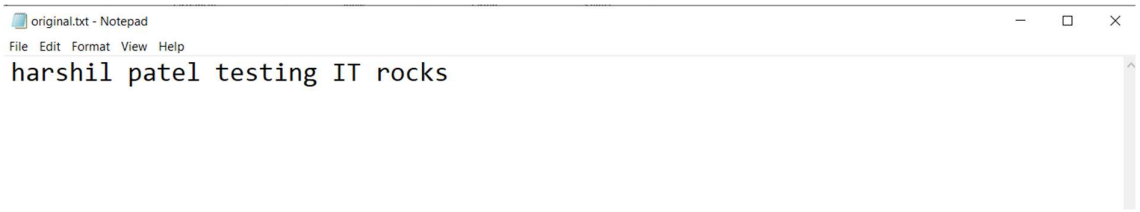
```
inFile.open("Binarycoded.txt");
OutFile.open("ANSIIcoded.txt");              //binary to ansiicode
int index=7;                                 //8 bit so start with 2^7
int sum=0;
cout<<endl<<"ANSII Values : ";
while(inFile>>text)
{
      if(text=='1')                    //1 mean add 2^index
            sum+=pow(2,index);
      index--;           //decrement index (2^7 next loop will be 2^6)
      if(index==-1)      //if the 8 bit checked than take another 8 bit
      {
            index=7;                          //reset to 7 index
            cout<<sum<<" ";
            char ch=sum;       //convert the value into ansii character
            OutFile<<ch; //write corresponding ansii character in the file
            sum=0;
      }
}
if(index!=-1)  //if len%8!=0 then convert the last sum generated into ansii
{
      char ch=sum;
      OutFile<<ch;
}
inFile.close();
OutFile.close();

inFile.open("ANSIIcoded.txt");
OutFile.open("Binarydecoded.txt");
while(inFile>>text)
{
      int v=text;
      v+=256;        //add 256 for gaining the desired number
      if(v!=256)     //last space has 256 so we will block by using if
```

```
            {
            string str=getbinary(v); //get binary value of number in string
            cout<<"\nValue "<<v;//<<"   Binary : "<<str;
            OutFile<<str;
            }
      }
      inFile.close();
      OutFile.close();

      inFile.open("Binarydecoded.txt");
      OutFile.open("output.txt");
      string matchit="";               //intial state
      while(inFile>>text)
      {
            matchit+=text;      //add the character which is read by infile
            int index=isfound(matchit,n);
            //return the index from the table which matches with the string
            if(index!=-1)       //-1 means value not available in the table
            {
                  OutFile<<data[index].ch;
                   //write corresponding index character into the file
                  matchit="";    //clear(reset) the matchit for next string
            }
      }
      return 0;
}
```

OUTPUT:



Binarycoded.txt - Notepad

File Edit Format View Help

```
01110101010000011110001001111110100010110100100101100000011000111011101110111000
11100101011110110101110000000
```

ANSIIcoded.txt - Notepad

File Edit Format View Help

```
uAâ~‹I` c»¼¯k€
```

Binarydecoded.txt - Notepad

File Edit Format View Help

```
01110101010000011110001001111110100010110100100101100000011000111011101110111000
11100101011110110101110000000
```

output.txt - Notepad

File Edit Format View Help

```
harshilpateltestingITrocks
```

C:\CODING\SEM 6 IT\DCDR\Practical 1 Shennon Fenon\testshenon.exe

```
KEY     OCCURENCE
I       1
T       1
a       2
c       1
e       2
g       1
h       2
i       2
k       1
l       2
n       1
o       1
p       1
r       2
s       3
t       3

Partition at index : 4
Partition at index : 1
Partition at index : 2
Partition at index : 8
Partition at index : 6
Partition at index : 11
Partition at index : 9
Partition at index : 13

Character       Frequency       Code
s               3               000
t               3               001
a               2               010
e               2               0110
h               2               0111
i               2               1000
l               2               1001
r               2               1010
I               1               1011
T               1               1100
c               1               11010
g               1               11011
k               1               11100
n               1               11101
o               1               11110
p               1               11111

Encoded Binary Text : 01110101010000011110001001111110100010110100100101100000011000111011101110111000101011110110101110000
ANSII Values : 117 65 226 126 139 73 96 99 187 188 175 107
--------------------------------
Process exited after 2.681 seconds with return value 0
Press any key to continue . . .
```