```
#include <iostream>
#include <cmath>
using namespace std;
struct node
{
   char info;
   int freq;
   char *code;
   node *Llink;
   node *Rlink;
};

class BinaryTree
{
  private:
      node *root;
      public:
          BinaryTree() { root=NULL; }
          void print();
          void assign_code(int i);
          void print_code(char c);
          void encode(const char str[]);
          void print_symbol(char cd[], int &f, int length);
          void decode(char cd[], int size);
      friend class minHeap;
      friend class HuffmanCode;
};

class minHeap
{
    private:
      BinaryTree *T;
      int n;
      public:
        minHeap();
        void heapify(int i);
        BinaryTree remove();
        void insert(BinaryTree b);
        void print();
        friend class HuffmanCode;
};

class HuffmanCode
{
    private:
        BinaryTree HuffmanTree;
        public:
            HuffmanCode();
};

HuffmanCode::HuffmanCode()
{
      minHeap Heap;
            while (Heap.T[0].root->freq>1)
      {
            BinaryTree l=Heap.remove();
            cout<<"\nAfter removing "<<l.root->freq<<endl;
            Heap.print();
            BinaryTree r=Heap.remove();
            cout<<"\nAfter removing "<<r.root->freq<<endl;
            Heap.print();
            HuffmanTree.root=new node;
            HuffmanTree.root->info='\0';
            HuffmanTree.root->freq=l.root->freq + r.root->freq;
            HuffmanTree.root->Llink=l.root;
            HuffmanTree.root->Rlink=r.root;
            cout<<"\nAfter inserting "<<l.root->freq<<"+"<<r.root->freq<<"=
"<<HuffmanTree.root->freq<<endl;
            Heap.print();
      }
      cout<<"\nThe process is completed and Huffman Tree is obtained\n";
      system ("pause");
      HuffmanTree=Heap.T[1];
      delete []Heap.T;
      cout<<"Traversal of Huffman Tree\n\n";
      HuffmanTree.print();
      system ("pause");
      cout<<"\nThe symbols with their codes are as follows\n";
      HuffmanTree.assign_code(0);
      system ("pause");
      cout<<"Enter the string to be encoded by Huffman Coding: ";
      char *str;
      str=new char[50];
      cin>>str;
      HuffmanTree.encode(str);
      system ("pause");
      int length;
      cout<<"Enter the code to be decoded by Huffman Coding: ";
      char *cd;
      cd=new char[60];
    cin>>cd;
      cout<<"Enter its code length: ";
      cin>>length;
      HuffmanTree.decode(cd,length);
      system ("pause");
}

minHeap::minHeap()
{
      cout<<"Enter no. of symbols:";
  cin>>n;
      T= new BinaryTree [n+1];
      T[0].root=new node;
      T[0].root->freq=n;
      for (int i=1; i<=n; i++)
  {
      T[i].root=new node;
                  cout<<"Enter characters of string :- ";
      cin>>T[i].root->info;
      cout<<"and their frequency of occurence in the string:- ";
      cin>>T[i].root->freq;
      T[i].root->code=NULL;
      T[i].root->Llink=NULL;
                  T[i].root->Rlink=NULL;
  }
  cout<<endl;
      int i=(int)(n / 2);
      cout<<"\nAs elements are entered\n";
      print();
  while (i>0)
  {
      heapify(i);
      i--;
  }
  cout<<"\nAfter heapification \n";
  print();
}
int min(node *a, node *b)
{if (a->freq <= b->freq) return a->freq;       else return b->freq;}
void swap(BinaryTree &a, BinaryTree &b)
{BinaryTree c=a;        a=b;        b=c;}

void minHeap::heapify(int i)
{
    while(1)
    {
                        if (2*i > T[0].root->freq)
                        return;
                        if (2*i+1 > T[0].root->freq)
                        {
                              if (T[2*i].root->freq <= T[i].root->freq)
                              swap(T[2*i],T[i]);
                              return;
                        }
        int m=min(T[2*i].root,T[2*i+1].root);
        if (T[i].root->freq <= m)
        return;
        if (T[2*i].root->freq <= T[2*i+1].root->freq)
        {
                              swap(T[2*i],T[i]);
                              i=2*i;
                        }
        else
        {
                              swap(T[2*i+1],T[i]);
                              i=2*i+1;
                        }
    }
}
BinaryTree minHeap::remove()
{
      BinaryTree b=T[1];
  T[1]= T[T[0].root->freq];
  T[0].root->freq--;
  if (T[0].root->freq!=1)
  heapify(1);
      return b;
}

void minHeap::insert(BinaryTree b)
{
      T[0].root->freq++;
      T[T[0].root->freq]=b;
      int i=(int) (T[0].root->freq /2 );
      while (i>0)
      {
            heapify (i);
            i=(int) (i /2 );
      }
}

int isleaf(node *nd)
{ if(nd->info=='\0') return 0; else return 1;}

void BinaryTree::assign_code(int i)
{
            if (root==NULL)
        return;
            if (isleaf(root))
            {
                  root->code[i]='\0';
                  cout<<root->info<<"\t"<<root->code<<"\n";
                  return;
            }
            BinaryTree l,r;
            l.root=root->Llink;
            r.root=root->Rlink;
            l.root->code=new char[i+1];
            r.root->code=new char[i+1];
            for (int k=0; k<i; k++)
            {
                  l.root->code[k]=root->code[k];
                  r.root->code[k]=root->code[k];
            }
            l.root->code[i]='0';
            r.root->code[i]='1';
            i++;
            l.assign_code(i);
            r.assign_code(i);
}

void BinaryTree::encode(const char str[])
{
      if (root==NULL)
    return;
      int i=0;
      cout<<"Encoded code for the input string '"<<str<<"' is\n";
      while (1)
      {
            if (str[i]=='\0')
            {
                  cout<<endl;
                  return;
            }
            print_code(str[i]);
            i++;
      }
}

void BinaryTree::print_code(char c)
{
      int f=0;
      if (isleaf(root))
      {
            if (c==root->info)
            {
```

```
                        f=1;
                        cout<<root->code;
                }
                return ;
        }
        BinaryTree l,r;
        l.root=root->Llink;
        if (f!=1)
        l.print_code(c);
        r.root=root->Rlink;
        if (f!=1)
        r.print_code(c);
}

int isequal(const char a[], const char b[], int length)
{
        int i=0;
        while (i<length)
        {
                if(b[i]!=a[i])
                return 0;
                i++;
        }
        if (a[i]!='\0')
        return 0;
        return 1;
}

void BinaryTree::decode(char cd[], int size)
{
        if (root==NULL)
  return;
        int i=0;
        int length=0;
        int f;
        char *s;
        cout<<"Decoded string for the input code '"<<cd<<"' is\n";
        while (i<size)
        {
                f=0;
                s=&cd[i];
                while (f==0)
                {
                        length++;
                        print_symbol(s,f,length);
                }
                i=i+length;
                length=0;
        }
        cout<<endl;
}

void BinaryTree::print_symbol(char cd[], int &f, int length)
{
        if (isleaf(root))
        {
                if (isequal(root->code, cd, length))
                {
                        f=1;
```

```
                  cout<<root->info;
            }
            return;
      }
      BinaryTree l,r;
      l.root=root->Llink;
      if (f!=1)
      l.print_symbol(cd,f,length);
      r.root=root->Rlink;
      if (f!=1)
      r.print_symbol(cd,f,length);
}

void BinaryTree::print()
{
  if (root==NULL)
  return;
  cout<<root->info<<"\t"<<root->freq<<"\n";
      if (isleaf(root))
      return;
      BinaryTree l,r;
      l.root=root->Llink;
      r.root=root->Rlink;
      l.print();
      r.print();
}

int power(int i, int j)
{
      int n=1;
      for (int k=1; k<=j; k++)
      n=n*i;
      return n;
}

int ispowerof2(int i)
{
      if (i==1)
      return 0;
      if (i==0)
      return 1;
      while (i>2)
      {
            if (i%2!=0)
            return 0;
            i=i/2;
      }
      return 1;
}

int fn(int l)
{
      if (l==1||l==0)
      return 0;
      return 2*fn(l-1)+1;
}

void minHeap::print()
{
      cout<<"The Heap showing the root frequencies of the Binary Trees are:\n";
      if (T[0].root->freq==0)
      {
            cout<<endl;
            system ("pause");
            return;
      }
      int level=1;
      while( T[0].root->freq >= power(2,level) )
      level++;
      if(level==1)
      {
            cout<<T[1].root->freq<<"\n";
            system ("pause");
            return;
      }
      for (int i=1; i<=T[0].root->freq; i++)
      {
            if (ispowerof2(i))
            {cout<<"\n"; level--;}
            for (int k=1; k<=fn(level); k++)
            cout<<" ";
            cout<<T[i].root->freq<<" ";
            for (int k=1; k<=fn(level); k++)
            cout<<" ";
      }
      cout<<endl;
      system ("pause");
}
int main()
{
    HuffmanCode c;
    system ("pause");
    return 0;
}
```