Mini-Projet
Compression

Ce sujet concerne la compression de fichiers texte. L'objectif est de transformer un fichier texte d'une taille N donnée en un autre fichier de taille M avec M<N (N et M sont respectivement) les nombres d'octets des fichiers d'origine et compressé). Le programme doit être réversible, c'est-à-dire qu'on doit pouvoir à partir du fichier compressé retrouver le fichiers d'origine.

>> **Compression -c source.txt destination.bin**
>> **Compression -d destination.bin source_bis.txt**

l'option -c indique au programme de procéder à une compression du fichier **source.txt.** Le résultat doit être stocké dans **destination.bin.**

l'option -d indique au programme de procéder à une décompression du fichier **destination.bin.** Le résultat doit être stocké dans **source_bis.txt**

Après ces commandes les fichiers **source_bis.txt** et **source.txt** doivent être identiques et la taille du fichier **destination.bin** est inférieure à celle de **source.txt.**

Méthode :
Idée générale :
Les données à encoder sont sous la forme d'une chaîne de caractères. Le principe du codage de Huffman est d' exploiter la fréquence de chacun des caractères, pour les représenter par quelque chose de plus adapté que leur code ASCII, qui lui est fixe. On associe ainsi un code plus court aux caractères les plus fréquemment rencontrés.

Exemple :
Soit S=«aaaaaaaaaabbbbbbbbc » la chaîne dont on veut optimiser la codage, c'est-à-dire minimiser le nombre de bits permettant de la stocker dans un fichier, de la lire à partir d'un fichier...

Caractères	a	b	c
effectif	10	8	1
code	00	01	1

Avec le codage ci-dessus, nous arrivons à 37 bits pour coder la séquence de caractères

Caractères	a	b	c
effectif	10	8	1
code	1	01	00

Avec le codage ci-dessus, nous arrivons à 28 bits pour coder la séquence de caractères.

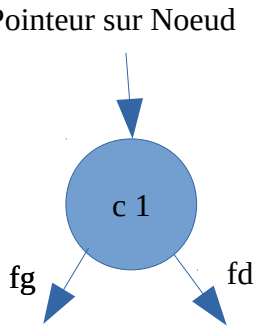
En effet « c » est le caractère le moins fréquent, c'est donc à celui-ci que nous devions associer le code le moins court.

Un autre point très important : si le code est mal choisi, exemple (a :1) et (b:11), le code «11» n'indique pas s'il faut le transformer en «a a» ou «b» . Il faut donc absolument que notre codage soit «préfixe», c'est à dire qu'il faut qu'aucun code ne soit préfixe d'un autre. Dans l'exemple (a:1) et préfixe de (b:11).

Solution : utiliser un arbre binaire dont les feuilles sont les caractères, et ou les arcs vers les fils sont étiquetés par des 0 en destination du fils gauche, et par un 1 en destination du fils droit. Le code d' un caractère est alors donné par des étiquettes des arcs de la racine jusqu'à la feuille étiquetée par ce caractères.

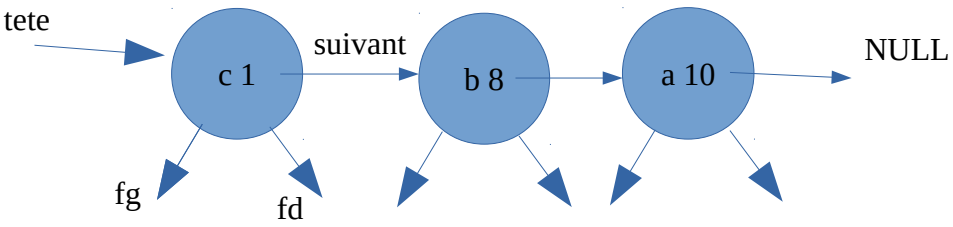
Un arbre binaire est un ensemble de Nœuds, chaque nœud contient une étiquette (les feuilles contiennent les caractères à coder), un entier correspondant à l'effectif associé au Nœud, un pointeur vers un Nœud gauche et un pointeur vers un Nœud droit.

```
class Noeud
{
        char c;
        int effectif ;
        Noeud *fg ;
        Noeud *fd ;
        ….
}
```

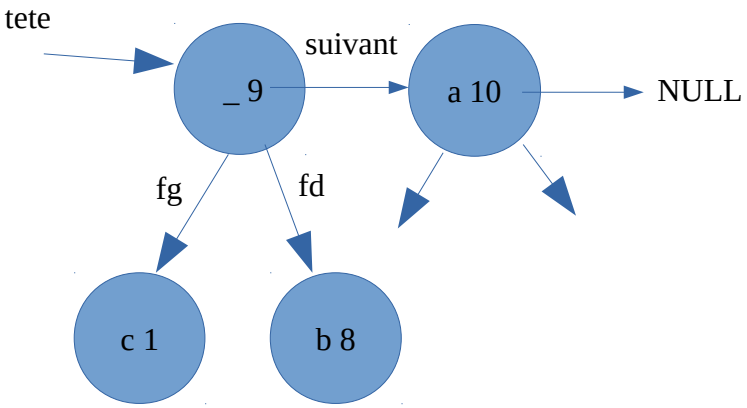


Tout d'abord on met tous les nœuds correspondants aux caractères de la séquence à coder dans des Nœuds feuilles. Nous décidons de stocker les Nœuds dans une liste chaînée, c'est la raison pour laquelle on rajoute à la classe Nœud un pointeur sur le Nœud suivant.

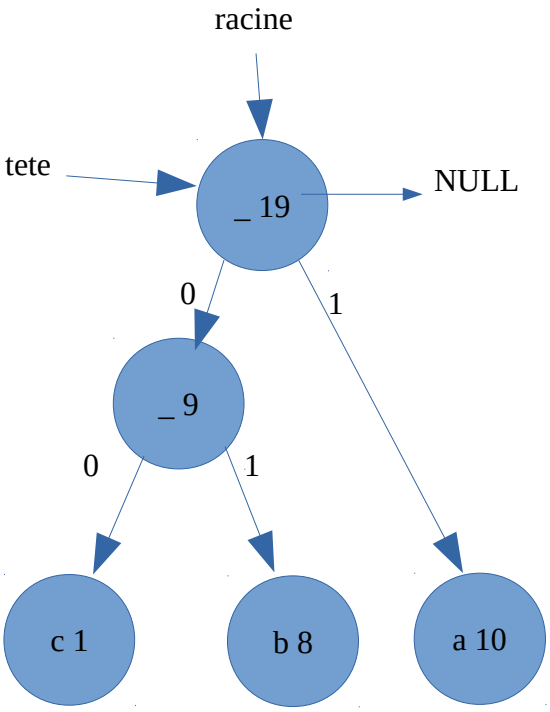
```
 class Noeud
{
        friend class arbre ;
        friend classe liste ;
        char c;
        int effectif ;
        Noeud *fg ;
        Noeud *fd ;
        Noeud *suivant ;
        ….
}
```



On détermine ensuite les deux Nœuds ayant les effectifs les plus faibles ; ici les Nœuds contenant les caractères « c » et « b ». Ensuite on crée un Nœud dont le fils gauche et droit sont les Nœuds associes à « c » et à « b » :



Les deux nœuds restant deviennent le fils gauche et le fils droit d'un nouveau nœud dont l'effectif est la somme des effectifs.



En parcourant l'arbre de haut en bas, ont voit que « c » est codé avec «00 », « b » avec « 01 » et « a » avec « 1 »

On considère la classe Noeud :
Soit la classe Noeud :

```
class Noeud
{
     friend class arbre;
     friend class liste ;
     int occ;
     char c;
     Noeud *suivant ;
     Noeud *fg;
     Noeud *fd;
     public:
     Noeud(const char a_c='\0',int a_occ=0,Noeud *a_suiv,\
                    Noeud *a_fg=NULL, Noeud *a_fd=NULL)
     {
          c=a_c;
          occ=a_occ;
          suivant=a_suiv ;
          fg=a_fg;
          fd=a_fd;
     }
}
```

On considère la classe liste :

```
class liste
{
     Noeud *tete;
     int nb_noeuds ;
     public:
     liste()
     {
          tete=NULL;
          nb_noeud=0 ;
     }
     void inserer_tete(Noeud *n) ;
     void inserer_tete(char c, int effectif) ;
     void inserer_les_caracteres(char *s, int taille) ;
     Noeud * supprimer_plus_petit() ;
     Noeud * la_tete(){return tete;}
     char *readfile(const string & filename, int & taille) ;
```

```
}
```

0 :

Lecture du fichier :
Dans la classe Liste écrire la fonction ***readfile*** qui qui prend en argument un nom de fichier et qui renvoie un tableau de caractères et un entier représentant la taille de ce tableau.

*char *Liste ::readfile(const string & filename, int & taille) ;*

```
char *readfile(const string & filename, int & taille)
{
     // création d'un objet de type ifstream :
     // grâce à f on pourra manipuler le fichier
     ifstream f(filename);
     // Aller à la find du fichier
     f.seekg(0,ios::end);
     // taille est le nombre d'octect dans le fichier
     taille=(int)f.tellg()-1;
     // allouer un tableau de la bonne taille
     char *s=new char[taille];
     // se replacer au début du fichier
     f.seekg(0,ios::beg);
     // lire tous les octects du fichier
     f.read(s,taille);
     // fermer le fichier
     f.close();
     // retourner le tableau ; retour de taille par variable
     return s;
}
```

1 :
- Ecrire les fonctions inserer_tete qui insère en tête de liste
- Ecrire la fonction ***inserer_les_caracteres :***

void liste ::inserer_les_caracteres(char *s, int taille) ;

qui reçoit un tableau de caractères et sa taille , et qui insère dans la liste tous les Nœuds correspondants à tous les caractères ayant un effectifs > 0. Considérez que chaque caractère est codé sur un entier compris en 0 et 255 qu'on appelle le code ASCII.

Cette fonction doit d'abord stocker les effectifs de tous les caractères dans un tableau d'entier ***T*** de taille 256. Parcourir le tableau de caractères et pour chaque caractère ***c*** rencontré incrémenter de ***T[(int) c].***

Ensuite pour chaque caractère ayant un effectif > 0, créer le Nœud correspondant et l'insérer dans la liste.

2 :
Ecrire dans la classe liste la fonction

Noeud *supprime_plus_petit() ;

qui supprime le Noeud contenant le plus ptit effectif et qui renvoie un pointeur dessus. La fonction renvoie NULL si la liste est vide.

```
Soit la classe arbre :
class arbre
{
     Noeud *r;
     string codes[256];
     public:
          arbre()
          {
               r=NULL ;
          }
     …..
} ;
```

r est un pointeur sur la racine de l'arbre. Dans le tableau ***codes*** on stockera les codes (sous forme de string). Chaque caractère est défini par sont code ASCCII qui est aussi l'indice dans le tableau ***codes***. Pour afficher par exemple le code associé au caractère 'u' :

cout<<code[(int)'u'] ; // affichera par exemple « 01001 »

codes

…..	
a	000
b	0
c	01
d	
……..	

3 :
Dans la classe ***arbre***, écrire une fonction ***construire_arbre*** qui prend en argument une liste de Noeuds et qui renvoie un pointeur sur un Nœud. Ce dernier représente la racine de l'arbre de l'algorithme ci-dessus.

4 :
Dans la classe arbre écrire le constructeur qui prend en argument un nom de fichier texte et qui construit l'arbre de codage correspondant.

5 :
Dans la classe arbre ; soit la fonction :

```
void codage()
{
     char code[20];
     int size=0;
     r→visiter(code,size,codes) ;
}
```

Écrire la fonction visiter (récursive) de la classe Nœud qui permettra en parcourant l'arbre récursivement de remplir la table ***« codes »*** de la classe arbre.

6 :
Écrire, dans la classe ***arbre*** la fonction :

```
string & arbre ::codage(char *s, int N, double & taux_compression)
```

qui prend en argument un tableau de char et qui renvoie un string contenant la suite de 0 et de 1 correspondant au codage en binaire du tableau de caractères.

Exemple :
Pour l'exemple plus haut : s="aaaaaaaaaabbbbbbbbc", le string renvoyé contiendra "11111111110101010101010100". Avant codage s occupe 8xtaille(s)=160 bits, et après codage seulement 29 bits. Le taux de compression est de : ((160-29)/160)*100=81,8 %

Question 6-
Écrire une fonction qui prend en argument un string (des 0 et des 1) et qui renvoie un tableau de char contenant les bits correspondant (c'est pas très clair), voici un exemple

T : "01100010010010010 (16 caractères)
Résultat : 01100010010010010 (2 octects=16bits)

Vous pouvez fournir une implémentation avec les deux méthodes suivantes :

Utilisation de la classe « bitset »

```
#include <iostream>
#include <string>
#include <bitset>
using namespace std;

int main()
{
     bitset<8> X(0);

    X[6]=1 ;
```

```
    //permet de mettre 1 dans le 7 bit
    //partant de la droite (bit de poids faible)
    X[5]=1 ;
    X[0]=1 ;
    char c=X.u_long() ;
    cout<<c<<endl
}
```

Ce petit programme affichera la lettre « a » dont le code ASCII est 97. En fait X contient 01100001=20+25+26=97 qui est bien le code ASCII de « a ».

Utilisation des décalage et des opération bit à bit

```
int main()
{
     char c=0 ;
     int n=1 ;
     for(int i=0;i<8;i++)
     {
         if(i==6 || i==5 || i==0) c=(int)c|(int)n;
         // avec l'opération | on pose le 1 du n dans c
         n=n<<1 ;
         // cette opération décale
         // les bits vers la gauche, vers les poids forts
     }
     cout<<c<<endl ;
}
```

Ce petit programme affichera la lettre « a » dont le code ASCII 97 = 01100001

Remarque : si le nombre de bits n'est pas multiple de 8, il faut compléter le dernier avec un code non déchiffrable par l'arbre. Par exemple une partie du plus long code.