Codigo de Huffman

a. Crear un archivo de texto plano formado por la combinación de letras “a”,”b”,”c”,”d,”e” y “f” en las siguientes cantidades:

- Me borre el code de la creacion del archivo, pero el resultado esta en es archivo ***"file.txt"***

```
a 45
b 13
c 12
d 16
e 9
f 5
```

- Codigo para leer el archivo ***"file.txt"***

```
void readFile(string fileDirection)
{
  ifstream file(fileDirection);
  if (file.is_open())
  {
    char character;
    while (!file.eof())
    {
      file >> character;
      cout << character << endl;
    }
  }
  else
    cout << "No se pudo abrir el archivo" << endl;
  file.close();
};
```

b. Crear una estructura de datos como se muestra a continuación:

Disculpe miss personalize la estructura 💪

1. La estructura LinkedList sera mi estructura de almacenamiento de lista de nodos

```
struct Node
{
  char _character;
  int _frecuency;
  struct Node *next;
  struct Node *left, *rigth;
};

// Lista de nodos
struct LinkedList
{
  Node *begin;
  Node *end;
};

Node *createNode(char character, int frecuency)
{
  Node *newNode = new (Node);
  newNode->_character = character;
  newNode->_frecuency = frecuency;
  newNode->left = nullptr;
  newNode->rigth = nullptr;
  newNode->next = nullptr;
  return newNode;
}
```

c. Leer el archivo de texto plano creado en el paso 1 y crear una lista ordenada. Observe que la lista se encuentra ordenada por la frecuencia de aparición de los caracteres.

1. Antes de completar los demas actividades del punto C debemos tener en cuenta lo siguiente:

```
void insertar(LinkedList &three, int frecuency, char character)
{
  Node *newNode = createNode(character, frecuency);
  // Si no existe nuestra lista la creamos
  if (!three.begin)
    three.begin = newNode;
  // Conforme vayan añadiendo
  // elementos a la lista vamos
  //  actualizando el valor de end
  else
    (three.end)->next = newNode;
  three.end = newNode;
}
```

- Conforme se vaya leyendo el archivo, crear el nodo y almacenar las frecuencias de aparición.
  - La funcion ***insertNodeFromFile*** funcion abre un archivo que almacena nuestras frecuencias.

```
void insertNodeFromFile(LinkedList &three, string fileDirection)
{
  ifstream file(fileDirection);
  if (file.is_open())
  {
    char character;
    int frecuency{0};
    while (!file.eof())
    {
      file >> character;
      file >> frecuency;
      // cout << character << " " << frecuency << endl;
      insertar(three, frecuency, character);
    };
  }
  else
    cout << "No se pudo abrir el archivo" << endl;
  file.close();
}
```

- Aplicar un algoritmo de ordenación para obtener la lista ordenada en base a la frecuencia

```
// Funcion para ordenar de forma ascendente por prioridad
void sortByFrecuency(LinkedList &three)
{
  // Nodos temporales para hacer el cambio
  Node *aux1, *aux2;
  // Nodos temporales para almacenar los punteros de left y ritgh de nuestro nodos
  Node *aux_left, *aux_rigth;
  char character;
  int frecuency;

  aux1 = three.begin;
  // Haremos uso de un doble while
  while (aux1->next)
  {
    aux2 = aux1->next;

    while (aux2)
    {
      // Si aux1 es mayor a aux2, haremos un swap
      if (aux1->_frecuency > aux2->_frecuency)
      {
        // Nuestro nodos temporales almacenaran el valor de aux1
        frecuency = aux1->_frecuency;
        character = aux1->_character;
        aux_rigth = aux1->rigth;
        aux_left = aux1->left;
        // Nuestro nodo aux1 toma los valores de aux2
        aux1->_frecuency = aux2->_frecuency;
        aux1->_character = aux2->_character;
        aux1->rigth = aux2->rigth;
        aux1->left = aux2->left;
        // Nuestro nodo aux2 toma los valores de los nodos temporales
        aux2->_frecuency = frecuency;
        aux2->_character = character;
        aux2->rigth = aux_rigth;
        aux2->left = aux_left;
      }
```

```
      aux2 = aux2->next;
    }
    aux1 = aux1->next;
  }
}
```

Resultado :

```
Lista de nodos
-----------------
C: a - F: 45 -> C: b - F: 13 -> C: c - F: 12 -> C: d - F: 16 -> C: e - F: 9 -> C: f - F: 5 -> nullptr
        Lista de nodos
-----------------
C: f - F: 5 -> C: e - F: 9 -> C: c - F: 12 -> C: b - F: 13 -> C: d - F: 16 -> C: a - F: 45 -> nullptr
```

d. Diseñe un algoritmo de tal manera que forme un árbol de Huffman:

```
// Funcion donde creamos nuestro Arbol
void Huffman(LinkedList &list)
{
  /*Importante antes de llamar al la funcion Huffman recordar que nuestra lista esta ordenada y el nodo padre es la suma
  // Creamos nodos temporales
  Node *leftNode, *rigthNode, *fatherNode;
  leftNode = list.begin;          // Nodo izquierdo
  rigthNode = (list.begin)->next; // Nodo derecho
  fatherNode = list.begin;        // Nodo padre

  int quantity = leftNode->_frecuency + rigthNode->_frecuency;

  while ((list.begin)->next)
  {
    fatherNode = createNode(' ', quantity); // Se crea un nuevo nodo

    fatherNode->left = leftNode;
    fatherNode->rigth = rigthNode;
    fatherNode->next = rigthNode->next; // Nodo padre es enlazado a la lista enlazada

    // Retiramos a los nodos hijos de la lista
    leftNode->next = nullptr;
    rigthNode->next = nullptr;
    // Ahora nuesta inicio de la lista sera el nuevo nodo padre
    list.begin = fatherNode;
    // Ordenamos nuestra lista
    sortByFrecuency(list);
    // Si nuestro nodo de inicio tiene un siguiente nodo
    if ((list.begin)->next)
    {
      // Tomamos los valores de los dos menores y repetimos el proceso en el inicio del bucle while
      leftNode = list.begin;
      rigthNode = leftNode->next;
      quantity = leftNode->_frecuency + rigthNode->_frecuency;
    }
  }
  // Imprimimos el arbol
  printThree(list);
}
```

Resultado:

```
       Nodo: 100       Left: 45        Rigth: 55
       Nodo: 55        Left: 25        Rigth: 30
       Nodo: 25        Left: 12        Rigth: 13
       Nodo: 30        Left: 14        Rigth: 16
       Nodo: 14        Left: 5         Rigth: 9
```

g. Proponga y diseñe un algoritmo, en referencia al árbol de Huffman, que decodifique un archivo de texto plano.

```
// Funcion para desencriptar
void desencrypt(LinkedList &three, string letters)
{
  Node *current = three.begin;
  int i = 0;
  //While para decodifficar cada letra
  while (i < letters.size())
```

```
  {
    // Mientras tengamos el Node padre
    while (current->left && current->rigth)
    {
      //Si es 0 va a la iquierda
      if (letters[i] == '0')
        current = current->left;
      //Si es 1 va a la derecha
      if (letters[i] == '1')
        current = current->rigth;
      i++;
    }
    cout << current->_character; //Imprimimos los caracteres
    current = three.begin;       //Para volver al recorrer con el siguiente caracter nuestro nodo debe volver al inicio
  }
  cout << endl;
}
```

Resultado

```
CADENA 000=aaa
CADENA 100011001101=cafe
CADENA 110001001101=face
```