especificacion TABLA[CLAVES,VALORES_MODIFICABLES]

tipos:
  tabla

operaciones:
  tabla_vacía: -->tabla {constructora}
  insertar_tabla:clave valor tabla --> tabla {constructora}
  consultar: tabla clave -->p valor
  esta_clave: tabla clave --> bool
  eliminar : clave tabla --> tabla

falta al menos
-> es_tabla_vacía

ecuaciones:
  insertar_tabla(c,w,insertar_tabla(c,v,t)) = insertar_tabla(c,combinar(v,w),t)
  insertar_tabla(c,w,insertar_tabla(d,v,t)) = insertar_tabla(d,v,insertar_tabla(c,w,t)) <== c!=d
  esta_clave(tabla_vacía,c) = falso
  esta_clave(insertar(c,w,t),d) = c == d v esta_clave(t,c)
  consultar(t, c) = error <== !esta_clave(t,c)
  consultar(insertar(d,w,t),d) =w <== !esta_clave(t,d)
  consultar(insertar(d,w,t),d) =combinar(w,consultar(d,w)) <== esta_clave(t,d)
  consultar(insertar(c,w,t),d) = consultar(t,d) <== c!=d
  eliminar(c,tabla_vacia) = tabla_vacia
  eliminar(c,insertar(c,v,t)) = eliminar(c,t)
  eliminar(c,insertar(d,w,t)) = insertar(d,w,eliminar(c,t))

especificación ARBOLES_HUFFMAN[CLAVES]

usa: BOOLEANOS, TABLA[CLAVES,CODIGOS_HUFFMAN], LISTA[CLAVES]

tipos:
  arbol_h

operaciones:
  hoja: clave nat -> arbol_h {constructora}
  plantar: arbol_h arbol_h ->p arbol_h {constructora}
  cto_elementos: arbol_h -> cto[clave]
  tabla_codigos: arbol_h -> tabla[clave,codigo_h]
  pre_cero: tabla[clave,codigo_h] --> tabla[clave,codigo]
  pre_uno: tabla[clave,codigo_h] --> tabla[clave,codigo_h]
  combinar_tablas: tabla[clave,codigo_h] tabla[clave,codigo_h] --> tabla[clave,codigo_h]
  decodifica_elem: codigo_h arbol_h -->p clave
  resto_dec : codigo_h arbol_h -->p lista[clave]
  decodifica : codigo_h arbol_h -->p lista[clave]
variables:
  izdo,dcho:arbol_h
  c,d:clave
  v,w:valor
  f:nat
  cod:codigo_h
  t,t1:tabla[clave,codigo_h]
ecuaciones:
  plantar(izdo,dcho) = error <== not es_vacio(interseccion(cto_elementos(izdo),cto_elementos(dcho)))
  cto_elementos(hoja(c,f)) = aniadir(c,cto_vacio)
  cto_elementos(plantar(izdo,dcho)) = union(cto_elementos(izdo),cto_elementos(dcho))
  pre_cero(insertar_tabla(c,v,t)) = insertar(c, 0:consultar(c,insertar_tabla(c,v,t))),eliminar(c,t))
  pre_uno(insertar_tabla(c,v,t)) = insertar(c, 1:consultar(c,insertar_tabla(c,v,t))),eliminar(c,t))
  combinar_tablas(tabla_vacia,t1) = t1
  combinar_tablas(insertar_tabla(c,v,t),t1) = insertar_tabla(c,v,combinar_tablas(t,t1))
  tabla_codigos(plantar(izdo,dcho)) = combinar_tablas(pre_cero(izdo),pre_uno(dcho))
  decodifica_elem(cod,hoja(c,f)) = c
  decodifica_elem(codigo_vacío,plantar(izdo,dcho)) = error
  decodifica_elem(0:cod,plantar(izdo,dcho)) = decodifica_elem(cod,izdo)
  decodifica_elem(1:cod,plantar(izdo,dcho)) = decodifica_elem(cod,dcho)
  resto_dec(cod,hoja(c,f)) = cod
  resto_dec(codigo_vacío,plantar(izdo,dcho)) = error
  resto_dec(0:cod,plantar(izdo,dcho)) = resto_dec(cod,izdo)
  resto_dec(1:cod,plantar(izdo,dcho)) = resto_dec(cod,dcho)
  decodifica(cod,a) = decodifica_elem(cod,a) + decodifica(resto_dec(cod,a),a)
  decodifica(codigo_vacío,a) = lista_vacía()

especificacion CODIGOS_HUFFMAN

tipos: codigo_h
operaciones:
  codigo_vacio : --> codigo_h
  _:_ : codigo_h codigo_h --> codigo_h
  0 : -->codigo_h
  1 : -->codigo_h
ecuaciones:
  codigo_vacio:c = c
  c:codigo_vacio = c
  (a:b):c = a:(b:c)

Notas de implementación de árbol_h:

Tenemos dos tipos de nodos:

- Las hojas con un elemento de tipo clave, sin hijos
- Los nodos internos (sin elemento asociado, con un hijo_izdo y un hijo_dcho)

Solución 1: sólo un tipo nodo_h con clave y dos hijos; si los dos hijos son NULL tenemos una hoja y si no. Si los dos hijos son distintos de NULL el campo clave no es relevante.

No hace falta nada nuevo de C++

Solución 2: dos tipos hoja, nodo_int que hereda de nodo_h

Precisa HERENCIA y métodos virtuales de los struct

```
struct nodo_h {
    ...
    virtual bool es_hoja() {
        return true;
    }
};
```

```
nodo_h* n = new nodo_int;
n -> es_hoja()
```

Llama a es_hoja() de nodo_int sin virtual llamaría a es_hoja() de nodo_h

```
struct hoja : nodo_h {
    ...
    bool es_hoja() {
        return true;
    }
};
```

```
struct nodo_int : nodo_h {
    ...
    bool es_hoja() {
        return false;
    }
};
```

Si a es un árbol_h (un puntero a Nodo_h)

El estilo de código que vamos a escribir es:

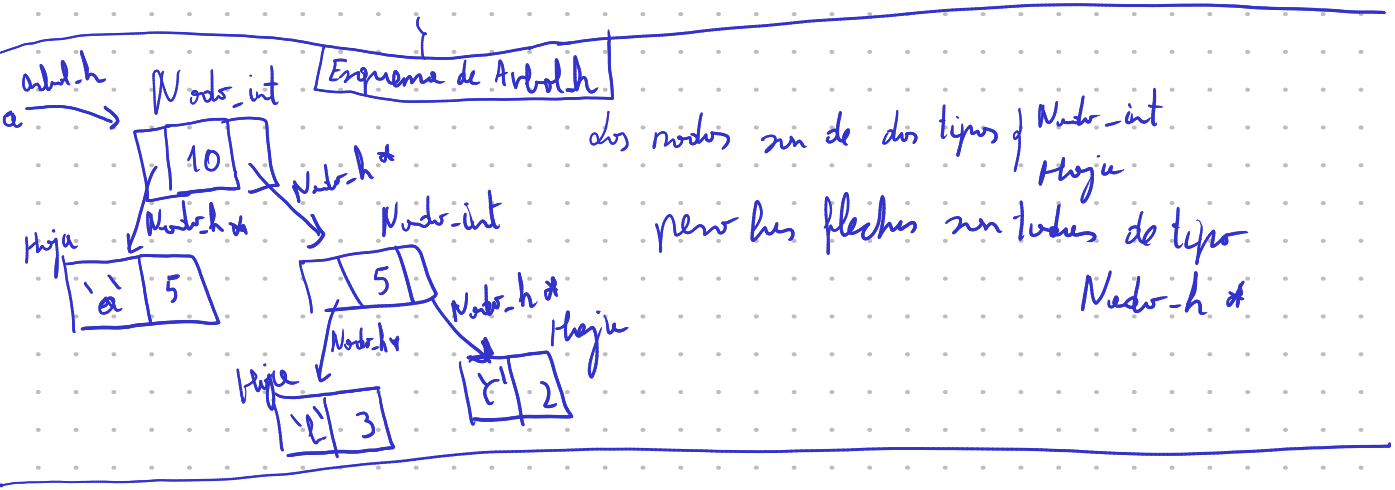
```
if(a->es_hoja()) {
    clave c = ((hoja*) a)->letra
```

sin la conversión a->letra no está bien pues a es de tipo Hoja

Si a es un arbol_h (un puntero a Nodo_h)
El estilo de código que vamos a escribir es:

```
if (a -> es_hoja() ){
    char c = ((Hoja *) a) -> letra
    . . . .
}
else{
    Arbol_h hijo = ((Nodo_h *) a) -> hijo_iz;
    . . . .
}
```

sin la conversión a->letra no estaría bien pues a es de tipo Nodo_h



dos nodos son de dos tipos { Nodo_int, Hoja
pero las flechas son todas de tipo Nodo_h*

Para el tipo Codigo_h usaremos vector<bool> (necesitamos #include <vector>)

El algoritmo para construir un Arbol_h a partir de una tabla de frecuencias usa una cola con prioridad de Arboles de Huffman. Usaremos priority_queue<arbol_h> (necesitamos #include <queue>)

```
priority_queue<Arbol_h> q;
```

→ necesita > para comparar arbol_h

Problema: C++ no deja redefinir > si no es un struct (o una clase). Dos soluciones →

- Definir
```
struct Arbol_h {
    Nodo_h * raiz;
};
```
en vez de `typedef Nodo_h * Arbol_h;`

Entonces ya podemos definir `bool operator>(Arbol_h h1, Arbol_h h2)`

- Definir
```
struct comp_Arbol {
    bool operator()(Arbol_h h1, Arbol_h h2) {
        return h1->fre > h2->fre;
    }
};
```

y la usa con:

```
comp_Arbol cmp;
cmp(h1,h2) → true si h1>h2   h1,h2 son Arbol_h
             false si no
```

```
priority_queue<Arbol_h, vector<Arbol_h>, comp_Arbol> q;
```

q.top() devuelve el mayor elemento de q
q.pop() elimina el mayor elemento de q
q.push(a) pone el arbol a en q

Para los códigos con `vector<bool>`

```
vector<bool> codigo;
codigo.push_back(true)   añade a la derecha
codigo.pop_back()        elimina a la derecha
codigo[i] → la posición i
codigo.size() → la longitud
```

vector no tiene implementado +. Podemos hacerlo con:

```
vector<bool> operator+(vector<bool> v1, vector<bool> v2){
    ...
}
```

Un ejemplo de main() para construir la tabla de códigos.

```
int main(){

    string file_name;
    cin >> file_name;
    ifstream f;
    f.open(file_name);
    tabla<char,int> t = tabla_vacia<char,int>();
    char c;
    int cont;
    while(!f.eof()){
      f>>c;
      aniadir(t,c,1);
      cont++;
    }
    f.close();
    Arbol_h a = arbol_h_from_frecs(t);
    tabla<char,cod_huffman> cods = tabla_vacia<char,cod_huffman>();
    tabla_codigos(a,cods,vector<bool>());
    cout << in_orden(cods);



  }
```