Manual Tecnico:

Proyecto: TablaHash
Lenguaje: C++
IDE : Visual Studio Code 1.41.1

Para la creacion de este proyecto se tuvo la implementacion de varias clases, de las cuales resaltan las de TablaHash Arbol y elemento las cuales son las que se encargan de almacenar la informacion de la empresa, lo que es el objetivo del proyecto.
Para poder insertar objetos en la TablaHash, el dato debe de pasar por funciones hash que devuelven una posicion en la tabla hash:
Funcion Hash Para Int: Es muy sencillo, simplemente se le saca el modulo al dato entero que se desea ingresar con el size de la tabla

```
int funcionHashInt(int dato, int size){
return (dato % size);}
```

Funcion Hash Para Float: al igual que el de int simplemente se le saca el modulo al dato float convertido a entero que se desea ingresar con el size de la tabla

```
int funcionHashFloat(float dato, int size){
return ((int)dato % size);
}
```

Funcion Hash Para String:se convierte los caracteres en una suma de ints para luego simplemente saca el modulo con el size de la tabla

```
int funcionHashString(string dato, int size)
{
int hashval, j;
hashval = (int)(dato.at(0));
for (j = 1; j < dato.size(); j++)
hashval += (int)dato.at(j);

return (hashval % size);
}
```

Funcion Hash Para Char: Es el mismo que el de string la diferencia es que no se recorre la parabra ya que es solo un carácter.

```
int funcionHashChar(char dato, int size)
{
int hashval = (int)(dato);
return (hashval % size);
}
```

Arbol AVL
Para el arbol avl se uso clases Nodo de la siguiente estructura:

```
class Nodo
{
public:
elemento *dato;
Nodo *izq, *dere, *padre;
int fe, alturaIzq, alturaDere, tipo;
private:
};
```

Y con estos nodos se hace mas facil la insertacion y los recorridos. Para insertar los nodos se comparaban el elemento del nodo a insertar y de los que ya existian:

```
void Arbol::insertarNuevo(Nodo *recorrer, Nodo *nuevo, Nodo *PadreAB)
{
bool vacio=false;
try{
if (raiz == NULL)
{
raiz = nuevo;
raiz->padre = NULL;
}else{
vacio=true;}
}catch(const std::exception& e){vacio=true;}
if(vacio){
if (nuevo->dato->dato <= recorrer->dato->dato){
if (recorrer->izq != NULL){
PadreAB = recorrer->izq;
insertarNuevo(recorrer->izq, nuevo, PadreAB);
}else{
recorrer->izq = nuevo;
nuevo->padre = PadreAB;
return;}
}else if (nuevo->dato->dato > recorrer->dato->dato){
if (recorrer->dere != NULL){
PadreAB = recorrer->dere;
insertarNuevo(recorrer->dere, nuevo, PadreAB);
}
else{
recorrer->dere = nuevo;
nuevo->padre = PadreAB;
return;}
}
}
}
```

Luego de cada insert se verifica si tiene que ser balanciado, encontrando alturas del arbol de la siguiente manera:
En donde las funciones rotarDD y rotarII son las notaciones necesarias para balanciar el arbol.

```
void Arbol::necesidadEquilibrar(Nodo *recorrer){ //Con esta función analizo si es necesario equilibrar el árbol, esta función busca el valor -2 o 2.
if (recorrer != NULL) {
necesidadEquilibrar(recorrer->izq);
necesidadEquilibrar(recorrer->dere);
if ((recorrer->fe==2)||(recorrer->fe==-2)){
system("cls");
cout<<"Es necesario equilibrar el arbol"<<endl;
PadreAB=recorrer;
sHijo=PadreAB->dere;//Identificar Si es el caso RDD, Rotación Derecha, Derecha.
if ((PadreAB->fe>1)&&(sHijo->fe>0)){
cout<<"Rotacion derecha derecha."<<endl;
cout<<" Padre: "<<PadreAB->dato<<" Hijo: "<<sHijo->dato<<endl;
rotarDD();
altura(raiz);
return;//Identificar Si es el caso RDI, Rotación Derecha, Izquierda.
}else if((PadreAB->fe>1)&&(sHijo->fe<0)){
cout<<"Rotacion derecha izquierda."<<endl;
cout<<" Padre: "<<PadreAB->dato<<" Hijo: "<<sHijo->dato<<endl;
rotarDI();
altura(raiz);
return;
}else{
PadreAB=recorrer;
sHijo=PadreAB->izq;//Identificar Si es el caso RII, Rotación Izquierda, Izquierda.
if((PadreAB->fe<-1)&&(sHijo->fe<0)){
cout<<"Rotacion izquierda izquierda."<<endl;
cout<<" Padre: "<<PadreAB->dato<<" Hijo: "<<sHijo->dato<<endl;
rotarII();
altura(raiz);
return;//Identificar Si es el caso RID, Rotación Izquierda, Derecha.
}else if((PadreAB->fe<-1)&&(sHijo->fe>0)){
cout<<"Rotacion izquierda Derecha."<<endl;
cout<<" Padre: "<<PadreAB->dato<<" Hijo: "<<sHijo->dato<<endl;
rotarID();
altura(raiz);
return;
}
}
return;
}
}
}
```