ANEXO DE AYUDA (CONTINUACIÓN):

```
//el constructor de la clase empresa debe crear un array dinámico de tamaño inicial 10
//debe inicializar a 0 los contadores de clientes (ncli) y contratos (ncon)
//y debe inicializar la constante nmaxcli a 100 y la variable nmaxcon a 10
Empresa::Empresa():nmaxcli(100) {
    this->ncli=0;
    this->ncon=0;
    this->contratos=new Contrato *[10]; //inicialmente capacidad para 10 Contratos
    this->nmaxcon=10;
}

//el destructor debe, además de eliminar el array dinámico creado en el constructor,
//eliminar los objetos clientes y contratos apuntados por ambos arrays
Empresa::~Empresa() {
    for(int i=0; i<this->ncon; i++) { //primero elimino los objetos contratos
        delete this->contratos[i];
    }
    delete [] this->contratos; //luego elimino el array de punteros

    for(int i=0; i<this->ncli; i++) { //primero elimino los objetos contratos
        delete this->clientes[i];
    }
  //delete [] this->clientes; //ERROR el array clientes no es dinámico
}

//método auxiliar usado por el método crearContrato
int Empresa::altaCliente(Cliente *c) { //añade cliente apuntado por c al array clientes
    int pos=-1;                        //devuelve -1 si no cabe y la posición donde
    if (this->ncli<nmaxcli) {          //donde lo he metido si cabe
        this->clientes[this->ncli]=c;
        pos=this->ncli;
        this->ncli++;
    }
    else {
        cout << "Lo siento, el cupo de clientes esta lleno";
        pos=-1;
    }
    return pos;
}

//método auxiliar usado por el método crearContrato
int Empresa::buscarCliente(long int dni) const {  //si no existe devuelve -1 y si existe
                                                   //devuelve la posición del cliente
    //A IMPLEMENTAR POR EL ALUMNO...               //con ese dni en el array clientes
}

void Empresa::crearContrato() {  //EL ALUMNO DEBE TERMINAR DE IMPLEMENTAR ESTE METODO
    long int dni;
    int pos;
    cout << "Introduce DNI: ";
    cin >> dni;
    //supongo que hay un metodo buscarCliente(dni) que devuelve -1 si no existe y si
    //existe devuelve la posicion del cliente en el array this->cli
    pos=this->buscarCliente(dni); //OJO ESTE METODO HAY QUE IMPLEMENTARLO
    if (pos==-1) { //el cliente no existe y hay que darlo de alta
        int dia, mes, anio;
        char nombre[100];
        Cliente *c; //NO CREO NINGUN CLIENTE SINO SOLO UN PUNTERO A CLIENTE
        cout << "Introduce Nombre: ";
        cin >> nombre;
        cout << "Introduce Fecha (dd mm aaaa): ";
        cin >> dia >> mes >> anio;
        c=new Cliente(dni, nombre, Fecha(dia, mes, anio));
        pos=this->altaCliente(c); //OJO HAY QUE IMPLEMENTARLO
    }
    //viendo cuanto vale la variable pos sé si el cliente se ha dado de alta o no
    if (pos!=-1) { //el cliente existe o se ha dado de alta
        //PREGUNTAR QUE TIPO DE CONTRATO QUIERE Y LOS DATOS NECESARIOS
        //CREAR EL OBJETO CONTRATO CORRESPONDIENTE Y AÑADIR AL ARRAY
        //contratos UN PUNTERO A DICHO OBJETO
    }
}
```

```
void Empresa::cargarDatos() {
 Fecha f1(29,2,2001), f2(31,1,2002), f3(1,2,2002);
 this->clientes[0] = new Cliente(75547001, "Peter Lee", f1);
 this->clientes[1] = new Cliente(45999000, "Juan Perez", Fecha(29,2,2000));
 this->clientes[2] = new Cliente(37000017, "Luis Bono", f2);
 this->ncli=3;

 this->contratos[0] = new ContratoMovil(75547001, f1, 0.12, 110, "DANES"); //habla 110m a 0.12€/m
 this->contratos[1] = new ContratoMovil(75547001, f2, 0.09, 170, "DANES"); //habla 170m a 0.09€/m
 this->contratos[2] = new ContratoTP(37000017, f3, 250); //habla 250m (300m a 10€, exceso 0.15€/m)
 this->contratos[3] = new ContratoTP(75547001, f1, 312); //habla 312m (300m a 10€, exceso 0.15€/m)
 this->contratos[4] = new ContratoMovil(45999000, f2, 0.10, 202, "ESPAÑOL"); //habla 202m a 0.10/m
 this->contratos[5] = new ContratoMovil(75547001, f2, 0.15, 80, "DANES"); //habla 80m a 0.15€/m
 this->contratos[6] = new ContratoTP(45999000, f3, 400); //habla 400m (300m a 10€, exceso 0.15€/m)
 this->ncon=7;
}
```

APÉNDICE 1)
Consideraciones respecto al código del destructor, altaCliente y buscarCliente…

El código mostrado considera que los clientes y contratos están todos de forma consecutiva en los arrays clientes y contratos, es decir, a la hora de dar de alta tanto los clientes como los contratos los vamos colocando uno a continuación de otro en sus respectivos arrays (cosa lógica)… y de la misma manera, cuando damos de baja un cliente y un contrato, el hueco que ocupaba dicho elemento debe ser tapado (de no ser así, los códigos anteriores darían error).

Si optamos por no tapar los huecos cuando un contrato o un cliente es eliminado de su respectivo array, entonces los códigos anteriores no sirven ya que consideran que no hay huecos.

Por tanto si no tapamos los huecos lo que habría que hacer es lo siguiente:
1º) al eliminar un elemento del array en dicha posición se debe guardar un NULL
2º) al dar de alta un elemento debemos recorrer el array desde el principio y guardar dicho elemento en la primera celda que tenga un NULL

De la misma manera, el destructor debe recorrerse el array completo y eliminar sólo aquellas celdas que tengan un valor distinto a NULL (ya que las que tienen un NULL no tienen nada en su interior)

El alumno tiene plena libertad para decidir qué estrategia debe seguir a la hora de dar de alta y/o de baja los elementos del array y debe en función de la estrategia seguida hacer el código de forma correcta.

Como he indicado anteriormente, el código mostrado arriba considera que los clientes y contratos son almacenados de forma consecutiva en sus respectivos arrays cuando son dados de alta ***y que cuando un cliente o contrato es eliminado el hueco que ocupaba en su respectivo array es tapado***.

Nota: En el siguiente apéndice se explica *cómo tapar* los huecos

APÉNDICE 2)
Cuidado al eliminar elementos de un array…

En la clase Empresa tenemos que implementar los métodos `cancelarContrato()` y `bajaCliente()`, los cuales deben eliminar elementos de los arrays `contratos` y `clientes` respectivamente.

Cuando eliminamos un elemento de un array *dejamos un hueco* que debe ser *tapado* desplazando los elementos contiguos una posición hacia atrás (si queremos conservar el orden) o colocando el último añadido *en el hueco* dejado por el elemento eliminado (más rápido y fácil de codificar, aunque no conserva el orden original).

Si por ejemplo tenemos un array de enteros y eliminamos el elemento que ocupa la 2º posición debemos desplazar los siguientes una posición para *tapar el hueco* dejado

Ejemplo de eliminación de un elemento de un array

```
int t[6] = {1, 3, 9, 3, 3, 2};                    // [1, 3, 9, 3, 3, 2]
int nmax=6, n=6;

Si eliminamos el 2º elemento               → // [1,  , 9, 3, 3, 2]
tapamos el hueco desplazando los contiguos → // [1, 9, 3, 3, 2]

Código para eliminar el numero 3 del array t (solo elimina el primero que encuentra)

int i=0;
bool encontrado=false;
while (i<n && encontrado==false) { //n indica cuantos elementos hay en el array t
  if (t[i]==3) { //busco la celda donde está el 3
    encontrado=true;
    //para eliminarlo desplazo las celdas adyacentes una posición a la izquierda
    for(int j=i+1; j<n; j++)
      t[j-1]=t[j]; //el que estaba en la posicion j (i+1) lo pongo en la posicion j-1 (i)
                   //y eso lo hago con todos los que hay desde i+1 hasta <n
    n--; //MUY IMPORTANTE: DECREMENTO n YA QUE HEMOS ELIMINADO UN ELEMENTO
  }
  else
    i++;
}

Si quiero eliminar TODAS las apariciones del numero 3 tengo que realizar un esquema de
recorrido en vez de un esquema de búsqueda ya que puede aparecer más veces.

Si hago lo siguiente cometemos un "pequeño gran" error:

for (int i=0; i<n; i++) { //n indica cuantos elementos hay en el array t
  if (t[i]==3) { //busco la celda donde está el 3
    //para eliminarlo desplazo las celdas adyacentes una posición a la izquierda
    for(int j=i+1; j<n; j++)
      t[j-1]=t[j]; //el que estaba en la posicion j (i+1) lo pongo en la posicion j-1 (i)
                   //y eso lo hago con todos los que hay desde i+1 hasta <n
    n--; //MUY IMPORTANTE: DECREMENTO n YA QUE HEMOS ELIMINADO UN ELEMENTO
  }
}

A simple vista parece que el código está correcto... pero si lo ejecutamos sobre el
siguiente ejemplo:
int t[6] = {1, 3, 9, 3, 3, 2};                    // [1, 3, 9, 3, 3, 2]
int nmax=6, n=6;

El array final que queda es el siguiente:         // [1, 9, 3, 2]

Es decir, hay un 3 que se "olvida" eliminar… y esto va a ocurrir cada vez que haya dos 3
consecutivos...
```

```
El error es debido a que el bucle for externo siempre incrementa la i cuando lo correcto
es incrementar la i sólo cuando no se desplaza las celdas en el for interno. Si tengo
dos 3 consecutivos la condición del if será cierta y se desplazarán las celdas
adyacentes una posición a la izquierda, con lo que el siguiente 3 que estaba a
continuación del otro pasará a ocupar la posición i. Al incrementar la i en el bucle
externo nos saltamos ese 3 y no lo borramos.


Hay 2 soluciones:

1) incrementar la i solo si el if es falso
2) recorrer el array descendentemente (el for i debe ir de n-1 a 0 en vez de 0 a n-1)



Solucion 1) incrementar la i solo si el if es falso

int i=0;
while (i<n) { //n indica cuantos elementos hay en el array t
  if (t[i]==3) { //busco la celda donde está el 3
    //para eliminarlo desplazo las celdas adyacentes una posición a la izquierda
    for(int j=i+1; j<n; j++)
      t[j-1]=t[j]; //el que estaba en la posicion j (i+1) lo pongo en la posicion j-1 (i)
                   //y eso lo hago con todos los que hay desde i+1 hasta <n
    n--; //MUY IMPORTANTE: DECREMENTO n YA QUE HEMOS ELIMINADO UN ELEMENTO
  }
  else
    i++;
}



Solucion 2) recorrer el array descendentemente (el for i debe ir de n-1 a 0 en vez de 0 a n-1)


for (int i=n-1; i>=0; i--) { //n indica cuantos elementos hay en el array t
  if (t[i]==3) { //busco la celda donde está el 3
    //para eliminarlo desplazo las celdas adyacentes una posición a la izquierda
    for(int j=i+1; j<n; j++)
      t[j-1]=t[j]; //el que estaba en la posicion j (i+1) lo pongo en la posicion j-1 (i)
                   //y eso lo hago con todos los que hay desde i+1 hasta <n
    n--; //MUY IMPORTANTE: DECREMENTO n YA QUE HEMOS ELIMINADO UN ELEMENTO
  }
}

En ambos casos, al ejecutarlo sobre el siguiente ejemplo:
int t[6] = {1, 3, 9, 3, 3, 2};                    // [1, 3, 9, 3, 3, 2]
int nmax=6, n=6;

El array final que queda es el siguiente:         // [1, 9, 2]
```

Por otro lado, en la práctica los arrays no tienen enteros sino punteros a objetos de tipo `Cliente` o `Contrato` (los cuales han sido creados dinámicamente usando `new`).

Por tanto, al eliminar un elemento de la tabla, no basta con desplazar los adyacentes una posición hacia la izquierda sino que antes debemos eliminar con `delete` el elemento a borrar.

De esto modo, si quiero eliminar del array `clientes` el objeto que es apuntado desde la posición 6 debo hacer lo siguiente:

```
//eliminar el elemento 6 del array de clientes


  delete this->clientes[6]; //libero la memoria del objeto apuntado por clientes[6]
//para tapar el hueco dejado desplazo las celdas adyacentes una posición a la izquierda
  for(int j=6+1; j<this->ncli; j++)
    this->clientes [j-1]= this->clientes [j];
  this->ncli--; //MUY IMPORTANTE: DECREMENTO n YA QUE HEMOS ELIMINADO UN ELEMENTO
```