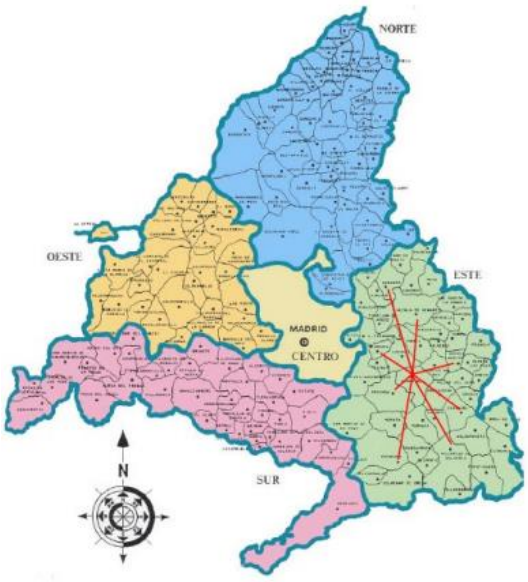
PECL 2- Estructuras de Datos

Simulación del funcionamiento de una red comarcal de paquetería



Creado por:

- Christian Noguerales Alburquerque – DNI: 06021665N
- Emilio Macías Do Santos – DNI: 53718519H

PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

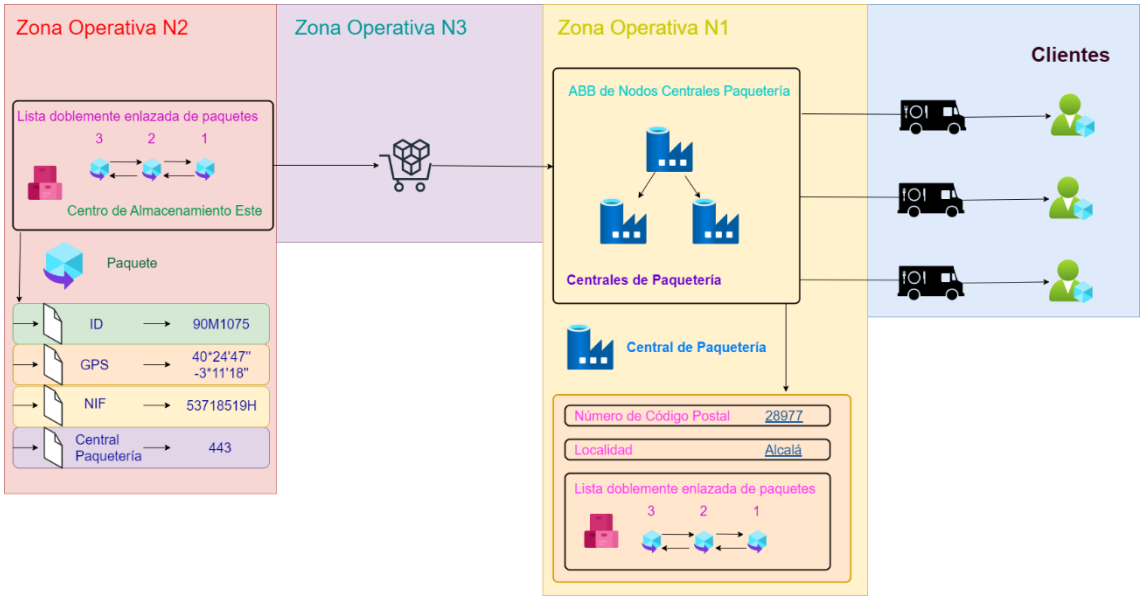
Contenido

PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

Portada de la memoria con el nombre, apellidos y DNI de quienes la hicieron.

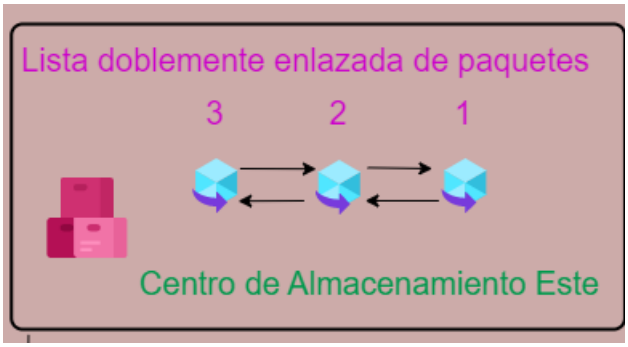
Dibujo de los TAD implementados, estructuras de datos definidas y cómo se relacionan entre ellos. Puede ser a mano, pero en ese caso la imagen insertada debe ser de buena calidad.

El proyecto tendría el siguiente planteamiento:



Tendremos las siguientes estructuras de datos:

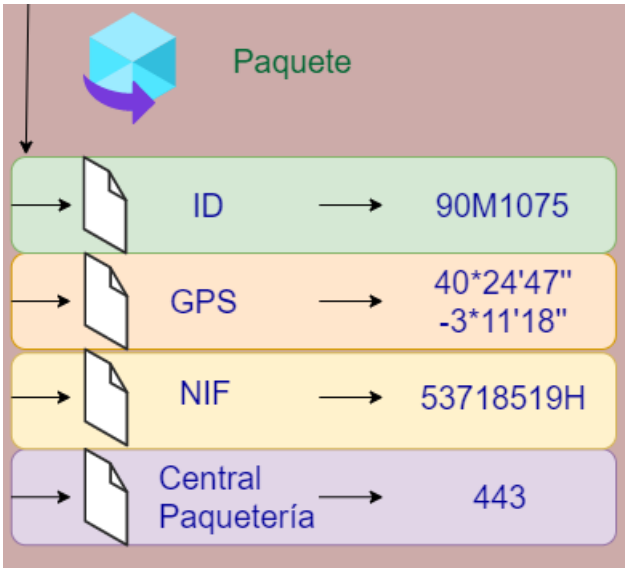
- Lista doblemente enlazada de Paquetes correspondiente a la Clase Centro de Almacenamiento Este.



  - Tiene comportamiento de Cola.
  - Contiene elementos de clase Paquete.

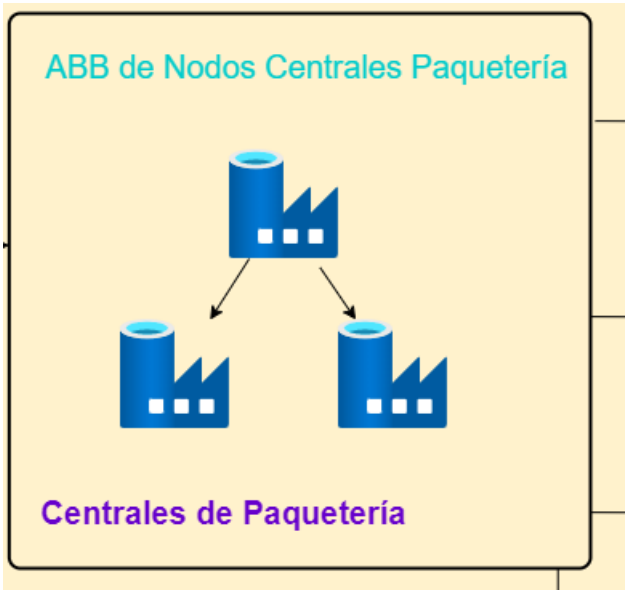
PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

- Clase Paquete:
  - Etiqueta identificativa del paquete con los siguientes campos (Estructura/Struct):
    1. ID (Identificador único del paquete). Se trata de un array de 7 elementos:
       - 2 números aleatorios
       - 1 char (carácter) aleatorio
       - 4 números secuenciales
       - Por ejemplo: 46D1018
    2. GPS (Lugar de destino del paquete) Se trata de una estructura (struct) con los siguientes elementos:
       - Latitud, array de 6 números
       - Longitud, array de 6 números
       - Por ejemplo: GPS:struct[latitud:int[6] = 403830, longitud:int[6] = 403310]
    3. NIF (ó DNI del cliente, sirve para identificar al cliente) Se trata de un array de 9 elementos
       - 8 de esos elementos son números
       - 1 elemento (el último elemento) es una letra de las siguientes [TRWAGMYFPDXBNJZSQVHLCKE]
    4. Número de Central de Paquetería de recepción.
       - Número entero de hasta 3 dígitos.
       - Por ejemplo: 234

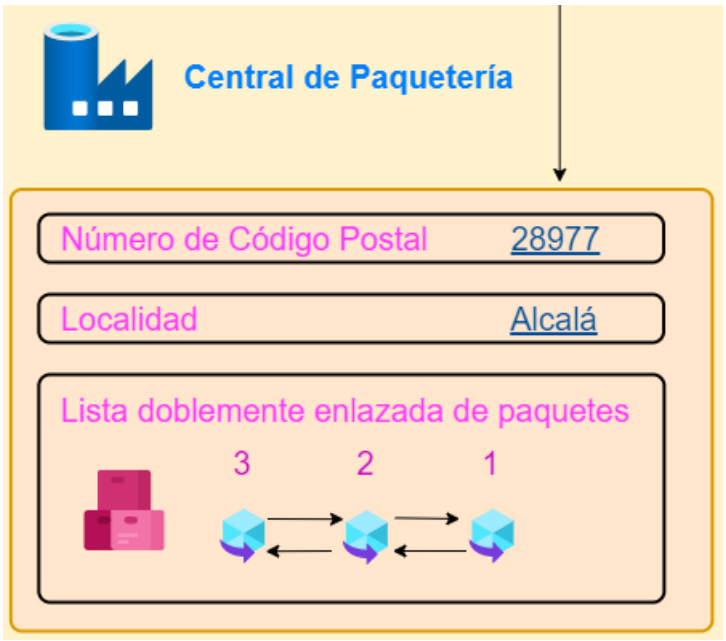


PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

- Arbol binario de búsqueda que almacena Centrales de Paquetería.



  - Clase Central de Paquetería:
    - Etiqueta identificativa de la Central de Paquetería con los siguientes campos (Estructura/Struct):
      1. Número de Código Postal de recepción:
         - Número entero de 5 dígitos.
         - Por ejemplo: 29077
      2. Localidad de recepción:
         - Objeto de clase String.
         - Por ejemplo: Alcalá
      3. Lista paquetes a entregar:
         - Lista doblemente enlazada de Paquetes a entregar.
         - Tiene comportamiento de cola.



PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

Habiendo definido anteriormente las estructuras de datos que utilizaremos en el proyecto pasamos a realizar la especificación de los mismos utilizando pseudocódigo.

Para esta práctica utilizaremos los siguientes TAD's:

Clase Paquete

La clase consta de los siguientes atributos:

-Etiqueta: struct

-ID: array[7], 2 num aleatorios,1 char (aleatorio),4 numeros secuenciales

-GPS: struct

-latitud:array[6]:numeros

-longitud:array[6]:numeros

-NIF: array[9], 8 son numeros + 1 letra

-Numero_CP: number, de hasta 3 dígitos; < 1000

Se ha determinado esta definición debido a que un paquete consta de una Etiqueta que identifica al paquete y de un conjunto de características, que dan información del paquete según el contexto sin llegar a identificarlo.

Quedando el siguiente pseudocódigo:

```
espec PAQUETE

usa Entero, Cadena

generos paquete

operaciones

var etiqueta_numero_cp, etiqueta_gps_latitud, etiqueta_gps_longitud:Entero

var etiqueta_id, nif:Cadena

fespec
```

Clase Central de Paquetería

Esta clase consta de los siguientes atributos:

-Numero_CP:int < 1000

-Localidad:String

-Lista_Paquetes de tipo lista doblemente enlazada.

Posee información sobre la localidad de entrega y un identificador. Lista_Paquetes almacena la cantidad de paquetes a entregar que sse corresponden a la misma zona de entrega. Definida por su localidad.

Quedando el siguiente pseudocódigo:

```
espec CentralPaqueteria

usa Entero, Logico, ListaDoble[Paquete]

generos centralpaqueteria

operaciones

var numero_cp:Entero

var lista_paquetes:ListaDoble[Paquete]

var localidad:String

fespec
```

Clase Cliente

La clase consta de los siguientes atributos:

-Datos:Struct

-NIF: array[9], 8 son numeros + 1 letra

Se ha determinado esta definición debido a que un cliente tiene una serie de datos identificadores (llave primaria y/o una/ninguna/varias llaves secundarias) para poder identificarlo y saber qué paquete le corresponde llegar a qué cliente.

¿Por qué se ha utilizado una estructura denominada Datos?

> Se ha utilizado la estructura Datos, debido a que lo que se encuentre dentro de esta estructura servirá para identificar a ese cliente sobre otros. Por ejemplo usando su NIF, su Nombre y Apellidos o incluso su correo electrónico si se desease. Utilizando esta estructuración de datos, nos garantizamos que si en un futuro deseamos añadir nuevos campos que identifiquen al cliente, podremos sin mucha complicación.

Si se añade un campo nuevo que no identifique al cliente, pero que sea importante guardarlo quedaría fuera de la estructura Datos, y se guardaría como un atributo directo en la clase Cliente.

Quedando el siguiente pseudocódigo:

> espec CLIENTE
>
> usa Cadena
>
> generos cliente
>
> operaciones
>
> var nif:Cadena
>
> fespec

Quedando la siguiente estructura de datos en PSeINT:

```
1  Algoritmo Cliente
2      Dimension nif[9]
3      Definir nif como Cadena
4  FinAlgoritmo
5
```

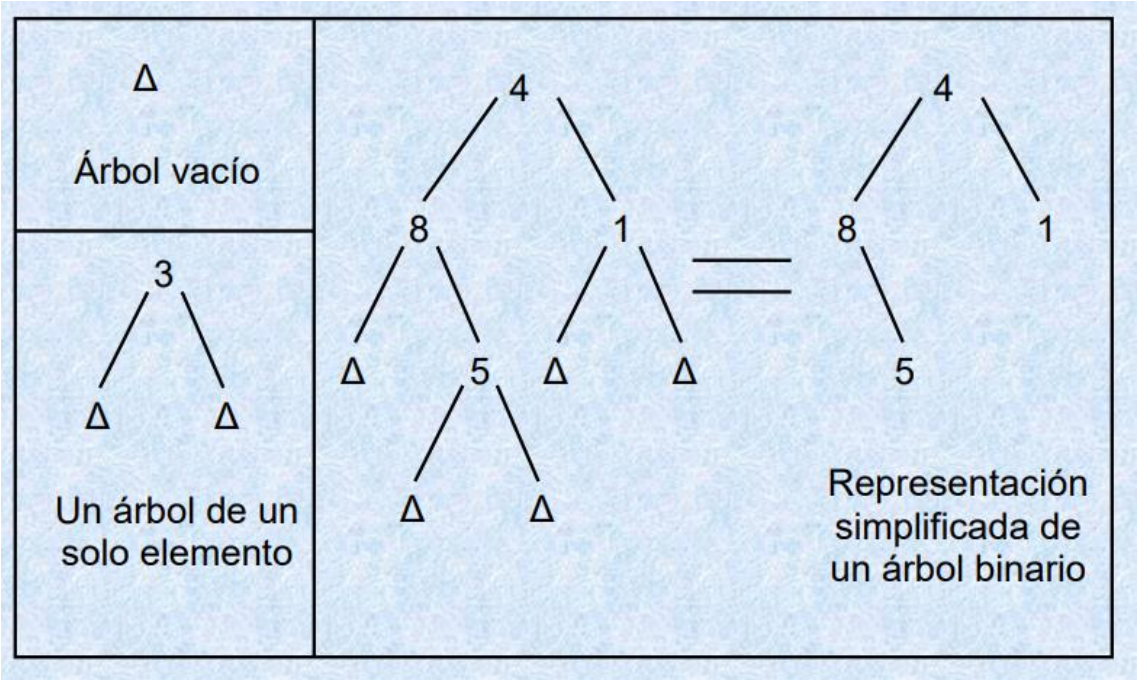
Estructura de Datos – CentralesPaqueteria

Como se ha explicado anteriormente, las Centrales de Paquería se tratan de nodos en un arbol binario de búsqueda. Por lo tanto, cuando se reciban los paquetes N3 de la lista doblemente enlazada del Centro de Almacenamiento Este, se irán guardando en el nodo correspondiente al Centro de Paquetería de la zona de entrega según se reciban.

Quedando el siguiente pseudocódigo:

```
espec CentralesPaqueteria[CentralPaqueteria]

usa NATURALES2, BOOLEANOS

parametro formal

generos Paquete

fparametro

generos arboles

fespec
```

Cabe destacar que a esta estructura de datos se le pasan EXCLUSIVAMENTE objetos de la clase Central Paquetería como parámetro, por lo que, esta estructura de datos se trata de un arbol binario donde solamente se puede usar para objetos de clase Central de Paquetería.



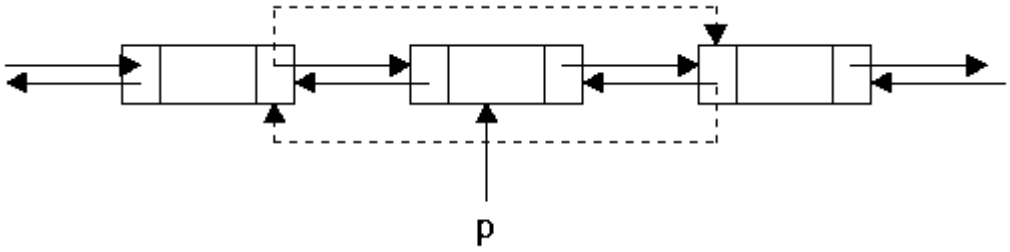
Estructura de Datos – CAE (Centro de Almacenamiento Este)

Como se ha explicado anteriormente, el Centro de Almacenamiento Este se trata de una lista doblemente enlazada, su estructura sería FIFO ó cola. Por lo tanto, cuando se almacenen temporalmente los paquetes N2 para su posterior entrega como paquetes N3 en las Centrales de Paquetería, los primeros que se hayan guardado serán los primeros en ser trasladados.

Quedando el siguiente pseudocódigo:

```
espec CAE[Paquete]

usa LISTA[Paquete], NATURALES2

parametro formal

generos Paquetes

fparametro

generos lista
```

Cabe destacar que a esta estructura de datos se le pasan EXCLUSIVAMENTE objetos de la clase Paquete como parámetro, por lo que, esta estructura de datos se trata de una lista donde solamente se puede usar para paquetes.



PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

Breve descripción de los métodos y funciones implementadas.

Se han definido las siguientes operaciones para tratar con las estructuras de datos especificadas anteriormente.

Estructura de Datos – CAE (Centro de Almacenamiento Este)

Para definir una lista doblemente enlazada, primero creamos sus nodos los cuales tendrán:

- Información a guardar, en este caso el paquete en cuestión.
- Nodo anterior, es un puntero al elemento anterior.
- Nodo siguiente, es un puntero al elemento siguiente.

```
//Lista
class NodoLista
{
private:
    Paquete paquete;
    NodoLista *siguiente;
    NodoLista *anterior;
    friend class Lista;

public:
    NodoLista(Paquete p, NodoLista *sig = NULL, NodoLista *ant = NULL)
    {
        paquete= p;
        siguiente = sig;
        anterior = ant;
    }
    //
    Paquete getPaquete(){
        return paquete;
    }
};
typedef NodoLista *pNodoLista;
```

Posteriormente creamos la clase Lista para terminar de definir esta estructura de datos:

```
class Lista
{
private:
    pNodoLista cabeza, actual, final;
public:
    Lista()
    {
        cabeza = actual = final = NULL;
    }
    ~Lista();
    void insertarNodo(Paquete p, char c);
    void borrarNodo(char c);
    pNodoLista borrarNodo();
    void borrarNodoIntermedio(pNodoLista nodo);
    bool listaVacia();
    void esCabeza();
    void esFinal();
    void esSiguiente();
    void esAnterior();
    bool esActual();
    Paquete paqueteActual();
    void mostrarLista();
    pNodoLista borrarAntiguo();
    int numNodos();
    pNodoLista existe(string idPaquete);
};
```

Destructor de la Lista:

```
//Métodos de tipos de datos:
Lista::~Lista()
{
    pNodoLista aux;
    esCabeza();
    while(cabeza)
    {
        aux = cabeza;
        cabeza = cabeza->siguiente;
        delete aux;
    }
    cabeza=NULL;
    actual = NULL;
    final=NULL;
}
```

Método para insertar un nuevo nodo a la lista:

```
void Lista::insertarNodo(Paquete p, char c)
{
    pNodoLista aux;
    char tipoInsercion;
    tipoInsercion=c;
    if (listaVacia()) // Si la lista está vacía
    {
        aux = new NodoLista(p,NULL,NULL);
        final = cabeza = aux;
    }
    else if (tipoInsercion=='f')  //Inserción por el final
    {
        aux= new NodoLista(p,NULL,NULL);
        aux->anterior=final;
        final->siguiente=aux;
        final = aux;
    }
    else if (tipoInsercion=='p')  //Inserción por el principio
    {
        aux= new NodoLista(p,NULL,NULL);
        aux->siguiente=cabeza;
        cabeza->anterior=aux;
        cabeza=aux;
    }
}
```

Método para borrar un nodo de la lista (distinguiendo entre borrar por el final o por el principio):

```
void Lista::borrarNodo(char c)
{
    char tipoBorrado;
    tipoBorrado=c;
    if(tipoBorrado=='f')  //Eliminación por el final
    {
        pNodoLista aux=NULL;
        if((cabeza==final))   //Sólo hay elemento
        {
            aux=final;
            cabeza = final = NULL;
            aux=NULL;
            delete aux;
        }
        else
        {
            aux=final;
            final=final->anterior;
            aux->anterior=NULL;
            final->siguiente=NULL;
            delete aux;
        }
    }

    else if(tipoBorrado=='p')  //Eliminación por el Principio
    {
        pNodoLista aux=NULL;
        if((cabeza==final))  //Sólo hay elemento
        {
            aux=cabeza;
            cabeza = final = NULL;
            aux=NULL;
            delete aux;
        }
        else
        {
            aux=cabeza;
            cabeza=cabeza->siguiente;
            aux->siguiente=NULL;
            cabeza->anterior=NULL;
            delete aux;
        }
    }
}
```

Método para borrar el nodo que se encuentre en la cabeza de la lista (el primer elemento) o borrar todos:

```
pNodoLista Lista::borrarNodo()
{
    // Si la lista está vacía tiene que devolver un NULL y no hacer nada
    if (listaVacia())
    {
        return NULL;
    }

    pNodoLista nodoBorrado = NULL;
    pNodoLista aux = cabeza;
    if (cabeza==actual)
    {
        if (actual==cabeza) actual=actual->siguiente;
        cabeza = actual->siguiente;
    }
    else
    {
        while (aux!=NULL && aux->siguiente!=actual)
        {
            aux = aux->siguiente;
        }
        aux->siguiente=actual->siguiente;
        if (aux->siguiente==NULL)
        {
            final=aux;
        }
    }
    nodoBorrado = actual;
    actual=actual->siguiente;
    nodoBorrado->siguiente=NULL;

    return nodoBorrado;
}
```

Método para eliminar un Nodo dado:

```
void Lista::borrarNodoIntermedio(pNodoLista nodo)
{

    // Si la lista está vacía tiene que devolver un NULL y no hacer nada
    if (listaVacia() || nodo==NULL)
    {
        return;
    }

    pNodoLista nodoBorrado = NULL;
    pNodoLista aux = cabeza;
    if (cabeza==nodo)
    {
        if (actual==cabeza) actual=nodo->siguiente;
        cabeza = nodo->siguiente;
    }
    else
    {
        while (aux!=NULL && aux->siguiente!=nodo)
        {
            aux = aux->siguiente;
        }
        aux->siguiente=nodo->siguiente;
        if (actual==nodo) actual = aux;
        if (aux->siguiente==NULL)
        {
            final=aux;
        }
    }
//    nodo->siguiente=NULL;
}
```

Método para mostrar los elementos de la lista:

```
void Lista::mostrarLista()
{
    pNodoLista aux;
    aux = cabeza;
    int j=0;
    while(aux!=NULL)
    {
        cout << setw(4)<< ++j << " " << setw(7) <<
             aux->getPaquete().idPaquete << " " << setw(9)
             << aux->getPaquete().NIF << " " << setw(2) <<
             aux->getPaquete().coordenadas.latitud[0] << "*"
             << setw(2) << aux->getPaquete().coordenadas.latitud[1]
             << "'" << setw(2) << aux->getPaquete().coordenadas.latitud[2]
             << "''" << " " << setw(2) << aux->getPaquete().coordenadas.longitud[0]
             << "*"<< setw(2) << aux->getPaquete().coordenadas.longitud[1] << "'" <<
             setw(2) << aux->getPaquete().coordenadas.longitud[2] << "''" << endl;
        aux = aux->siguiente;
    }
}
```

Métodos auxiliares:

```
void Lista::esCabeza()
{
    actual = cabeza;
}

void Lista::esFinal()
{
    actual=final;
}

void Lista::esSiguiente()
{
    if(actual) actual = actual->siguiente;
}

void Lista::esAnterior()
{
    if(actual) actual = actual->anterior;
}

bool Lista::esActual()
{
    return actual != NULL;
}

Paquete Lista::paqueteActual()
{
    return actual->paquete;
}
```

Método para contar el número de nodos de la lista (elementos):

```
int Lista::numNodos()
{
    pNodoLista aux;
    aux = cabeza;
    int j=0;
    while(aux!=NULL)
    {
        j++;
        aux = aux->siguiente;
    }
    return j;
}
```

Método para saber si existe un paquete buscando por id:

```
pNodoLista Lista::existe(string idPaquete)
{
    Paquete p;
    pNodoLista aux = cabeza;
    while (aux!=NULL)
    {
        p = aux->getPaquete();
        if (p.idPaquete==idPaquete)
        {
            return aux;
        }
        else
        {
            aux = aux->siguiente;
        }
    }
    return NULL;
}
```

Estructura de Datos – CentralesPaqueteria

Para definir la clase, se ha programado los siguientes métodos y funciones del arbol binario:

Se ha definido una clase NodoArbol para especificar los Nodos:

```
class NodoArbol {
    private:
        // Miembros:
        CP centralP;
        NodoArbol *izquierdo;
        NodoArbol *derecho;
        friend class Arbol;

    public:
        // Constructor:
        NodoArbol(CP cp, NodoArbol *izq=NULL, NodoArbol *der=NULL) :
        centralP(cp), izquierdo(izq), derecho(der) {};
        CP getCentrarlP(){
            return centralP;
        };

};
typedef NodoArbol *pNodoArbol;
```

Para terminar definir esta estructura de datos, se ha creado la clase Arbol:

```
class Arbol {
    private:
        // Punteros de la lista, para cabeza y nodo actual:
        pNodoArbol raiz;
        pNodoArbol actual;
        int contador;
        int altura;

    public:
        // Constructor y destructor básicos:
        Arbol() : raiz(NULL), actual(NULL) {}
        ~Arbol() { podar(raiz); }
        // Insertar en árbol ordenado:
        void insertar(CP cp);
        // Borrar un elemento del árbol:
        void borrar(CP cp);
        // Función de búsqueda:
        pNodoArbol buscar(CP cp);
        // Comprobar si el árbol está vacío:
        bool vacio(pNodoArbol r) { return r==NULL; }
        // Comprobar si es un nodo hoja:
        bool esHoja(pNodoArbol r) { return r->derecho!=NULL && r->izquierdo!=NULL; }
        // Contar número de nodos:
        const int numeroNodos();
        const int alturaArbol();
        // Calcular altura de un int:
        int calculaAltura(CP cp);
        // Devolver referencia al int del nodo actual:
        CP cpActual() { return actual->centralP; } //&ValorActual()
        // Moverse al nodo raiz:
        void volverARaiz() { actual = raiz; }
        // Aplicar una función a cada elemento del árbol:
        void inOrden(void (*func)(int&) , pNodoArbol nodo=NULL, bool r=true);
        void preOrden(void (*func)(int&) , pNodoArbol nodo=NULL, bool r=true);
        void postOrden(void (*func)(int&) , pNodoArbol nodo=NULL, bool r=true);
        void inOrdenCP(void (*func)(CP&) , pNodoArbol nodo=NULL, bool r=true);
        void preOrdenCP(void (*func)(CP&) , pNodoArbol nodo=NULL, bool r=true);
        void postOrdenCP(void (*func)(CP&) , pNodoArbol nodo=NULL, bool r=true);
        Paquete *buscarInOrdenCP(Paquete *(*func)(CP&, string),string idPaquete, pNodoArbol nodo=NULL, bool r=true);
        pNodoArbol buscarNodoInOrdenCP(bool (*func)(CP&, string),string idPaquete, pNodoArbol nodo=NULL, bool r=true);
        private:
        // Funciones auxiliares
        void podar(pNodoArbol &nodo);
        void auxContador(pNodoArbol nodo);
        void auxAltura(pNodoArbol nodo, int alt);
};
```

Para eliminar un nodo del arbol y sus hijos:

```
// Poda: borrar todos los nodos a partir de uno, incluido
void Arbol::podar(pNodoArbol &NodoArbol)
{
    // Algoritmo recursivo, recorrido en postorden
    if(NodoArbol)
    {
        podar(NodoArbol->izquierdo); // Podar izquierdo
        podar(NodoArbol->derecho);   // Podar derecho
        delete NodoArbol;            // Eliminar nodo
        NodoArbol = NULL;
    }
}
```

Para insertar un nuevo nodo:

```
// Insertar un int en el árbol ABB
void Arbol::insertar(CP cp)
{
    pNodoArbol padre = NULL;

    actual = raiz;
    // Buscar el int en el árbol, manteniendo un puntero al nodo padre
    while(!vacio(actual) && cp.numCP  != actual->centralP.numCP)
    {
        padre = actual;
        if(cp.numCP > actual->centralP.numCP) actual = actual->derecho;
        else if(cp.numCP < actual->centralP.numCP) actual = actual->izquierdo;
    }

    // Si se ha encontrado el elemento, regresar sin insertar
    if(!vacio(actual)) return;
    // Si padre es NULL, entonces el árbol estaba vacío, el nuevo nodo será
    // el nodo raiz
    if(vacio(padre)) raiz = new NodoArbol(cp);
    // Si el int es menor que el que contiene el nodo padre, lo insertamos
    // en la rama izquierda
    else if(cp.numCP < padre->centralP.numCP) padre->izquierdo = new NodoArbol(cp);
    // Si el int es mayor que el que contiene el nodo padre, lo insertamos
    // en la rama derecha
    else if(cp.numCP > padre->centralP.numCP) padre->derecho = new NodoArbol(cp);
}
```

Para eliminar un nodo del arbol, si no es directo lo vamos recorriendo hasta borrarlo:

```
// Eliminar un elemento de un árbol ABB
void Arbol::borrar(CP cp)
{
    pNodoArbol padre = NULL;
    pNodoArbol nodo;
    CP aux;

    actual = raiz;
    // Mientras sea posible que el valor esté en el árbol
    while(!vacio(actual))
    {
        if(cp.numCP == actual->centralP.numCP)    // Si el valor está en el nodo actual
        {
            if(esHoja(actual))    // Y si además es un nodo hoja: lo borramos
            {
                if(padre)  // Si tiene padre (no es el nodo raiz)

                    // Anulamos el puntero que le hace referencia
                    if(padre->derecho == actual) padre->derecho = NULL;
                    else if(padre->izquierdo == actual) padre->izquierdo = NULL;

                delete actual; // Borrar el nodo
                actual = NULL;
                return;
            }
            else   // Si el valor está en el nodo actual, pero no es hoja
            {
                // Buscar nodo
                padre = actual;
                // Buscar nodo más izquierdo de rama derecha
                if(actual->derecho)
                {
                    nodo = actual->derecho;
                    while(nodo->izquierdo)
                    {
                        padre = nodo;
                        nodo = nodo->izquierdo;
                    }
                }

                // O buscar nodo más derecho de rama izquierda
                else
                {
                    nodo = actual->izquierdo;
                    while(nodo->derecho)
                    {
                        padre = nodo;
                        nodo = nodo->derecho;
                    }
                }
                // Intercambiar valores de no a borrar u nodo encontrado
                // y continuar, cerrando el bucle. El nodo encontrado no tiene
                // por qué ser un nodo hoja, cerrando el bucle nos aseguramos
                // de que sólo se eliminan nodos hoja.
                aux = actual->centralP;
                actual->centralP = nodo->centralP;
                nodo->centralP = aux;
                actual = nodo;
            }
        }
        else   // Todavía no hemos encontrado el valor, seguir buscándolo
        {
            padre = actual;
            if(cp.numCP > actual->centralP.numCP) actual = actual->derecho;
            else if(cp.numCP < actual->centralP.numCP) actual = actual->izquierdo;
        }
    }
```

Para recorrer el arbol:

```
// Recorrido de árbol en inorden, aplicamos la función func, que tiene
// el prototipo:
// void func(int&);
void Arbol::inOrden(void (*func)(int&), pNodoArbol nodo, bool r)
{
    if(r) nodo = raiz;
    if(nodo->izquierdo) inOrden(func, nodo->izquierdo, false);
    func(nodo->centralP.numCP);
    if(nodo->derecho) inOrden(func, nodo->derecho, false);
}

// Recorrido de árbol en preorden, aplicamos la función func, que tiene
// el prototipo:
// void func(int&);
void Arbol::preOrden(void (*func)(int&), pNodoArbol nodo, bool r)
{
    if(r) nodo = raiz;
    func(nodo->centralP.numCP);
    if(nodo->izquierdo) preOrden(func, nodo->izquierdo, false);
    if(nodo->derecho) preOrden(func, nodo->derecho, false);
}

// Recorrido de árbol en postorden, aplicamos la función func, que tiene
// el prototipo:
// void func(int&);
void Arbol::postOrden(void (*func)(int&), pNodoArbol nodo, bool r)
{
    if(r) nodo = raiz;
    if(nodo->izquierdo) postOrden(func, nodo->izquierdo, false);
    if(nodo->derecho) postOrden(func, nodo->derecho, false);
    func(nodo->centralP.numCP);
}
```

Para buscar un valor en el arbol:

```
// Buscar un valor en el árbol
pNodoArbol Arbol::buscar(CP cp)
{
    actual = raiz;

    // Todavía puede aparecer, ya que quedan nodos por mirar
    while(!vacio(actual))
    {
        if(cp.numCP == actual->centralP.numCP) return actual; // int encontrado
        else if(cp.numCP > actual->centralP.numCP) actual = actual->derecho; // Seguir
        else if(cp.numCP < actual->centralP.numCP) actual = actual->izquierdo;
    }
    return NULL; // No está en árbol
}
```

Para calcular su altura:

```
// Calcular la altura del nodo que contiene el int dat
int Arbol::calculaAltura(CP cp)
{
    int altura = 0;
    actual = raiz;

    // Todavía puede aparecer, ya que quedan nodos por mirar
    while(!vacio(actual))
    {
        if(cp.numCP == actual->centralP.numCP) return altura; // int encontrado
        else
        {
            altura++; // Incrementamos la altura, seguimos buscando
            if(cp.numCP > actual->centralP.numCP) actual = actual->derecho;
            else if(cp.numCP < actual->centralP.numCP) actual = actual->izquierdo;
        }
    }
    return -1; // No está en árbol
}
```

Contar su número de nodos:

```
// Contar el número de nodos
const int Arbol::numeroNodos()
{
    contador = 0;

    auxContador(raiz); // FUnción auxiliar
    return contador;
}
```

Para mostrar el arbol por pantalla:

```
// Función mostrar el contenido de los nodos del árbol
void Mostrar(int &d)
{
    cout << d << ",";
}
```

Para generar el arbol:

```
//Función que permite generar árboles de forma aleatoria
void generarArbol(Arbol& arbolNew)
{
    int i = 0;
    int num=0;
    int MAX=20;

    for (i=0; i<MAX; i++)
    {
        num=rand()%MAX;
        cout << num << endl;
//      arbolNew.insertar(num); // Santi, aquí tendríais que generar un CP de forma aleatoria. No entiendo la utilidad de este método. Yo lo quitaría
    }
}
```

Funciones auxiliares:

```
// Función auxiliar para contar nodos. Función recursiva de recorrido en
//   preorden, el proceso es aumentar el contador
void Arbol::auxContador(pNodoArbol nodo)
{
    contador++;  // Otro nodo
    // Continuar recorrido
    if(nodo->izquierdo) auxContador(nodo->izquierdo);
    if(nodo->derecho)   auxContador(nodo->derecho);
}

// Calcular la altura del árbol, que es la altura del nodo de mayor altura.
const int Arbol::alturaArbol()
{
    altura = 0;

    auxAltura(raiz, 0); // Función auxiliar
    return altura;
}

// Función auxiliar para calcular altura. Función recursiva de recorrido en
// postorden, el proceso es actualizar la altura sólo en nodos hojas de mayor
// altura de la máxima actual
void Arbol::auxAltura(pNodoArbol nodo, int a)
{
    // Recorrido postorden
    if(nodo->izquierdo) auxAltura(nodo->izquierdo, a+1);
    if(nodo->derecho)   auxAltura(nodo->derecho, a+1);
    // Proceso, si es un nodo hoja, y su altura es mayor que la actual del
    // árbol, actualizamos la altura actual del árbol
    if(esHoja(nodo) && a > altura) altura = a;
}
```

PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

Estas son las estructuras de datos más importantes, a continuación se procederá a explicar el resto de clases del programa.

- Estructura Paquete

```
struct Paquete
{
    string idPaquete;
    CoordenadasGPS coordenadas;
    string NIF;
    int numCP;
};
```

Generamos paquetes de forma aleatoria:

```
string generarIdPaquete()
{
    string id;
    char abecedario[] = {'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};
    string parte1 = to_string(rand()%99);
    string parte2 = to_string(rand()%9999);
    id= padTo(parte1,2)+  abecedario[rand()%26] + padTo(parte2,4);
//    id = padTo(to_string(rand()%99),2) +  abecedario[rand()%26] + padTo(to_string(rand()%9999),4);

    return id;
}
```

```
string generarNIF()
{
    int numero;
    string NIF;
    char letras[] = {'T','R','w','A','G','M','Y','F','P','D','X','B','N','J','Z','S','Q','V','H','L','C','K','E'};

    for (int i=0; i<6; i++)
    {
        numero = numero*10 + rand()%10;
    }
    NIF = to_string(numero) + letras[numero%23];

    return NIF;
}
```

```
Paquete generarPaquete(int numCp)
{
    Paquete p;

    p.idPaquete = generarIdPaquete();
    p.coordenadas = generarCoordenadas();
    p.NIF = generarNIF();
    p.numCP = numCp;

    return p;
}
```

PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

- Estructura CoordenadasGPS

```
//Definición de tipos de datos:
struct CoordenadasGPS
{
    int latitud[3];
    int longitud[3];
};
```

Generamos las coordenadas de forma aleatoria:

```
CoordenadasGPS generarCoordenadas()
{
    CoordenadasGPS c;
    int gradLat, minLat, segLat, gradLong, minLong, segLong;

    gradLat = 40;
    minLat = 6 + rand()%28;
    segLat = rand()%60;
    gradLong = -3;
    minLong = 5 + rand()%29;
    segLong = rand()%60;

    c.latitud[0] = gradLat;
    c.latitud[1] = minLat;
    c.latitud[2] = segLat;
    c.longitud[0] = gradLong;
    c.longitud[1] = minLong;
    c.longitud[2] = segLong;

    return c;
}
```

PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

- Estructura Central de Paquetería

```
struct CP
{
    int numCP;
    string localidad;
    Lista *listaPaq;
};
```

Generamos sus atributos de forma aleatoria:

```
int generarNumCP()
{
    return rand()%9999;
}
string AsignarLocalidades()//Revisar
{
    int size= 13;
    int aleatorio=rand()%size;
    return localidades[aleatorio];
}
```

```
CP generarCpPredeterminado()
{
    CP cp;
    cp.numCP=generarNumCP();
    cp.localidad=AsignarLocalidades();
    cp.listaPaq=new Lista();

    return cp;
}
```

Métodos para ver sus valores:

```
void muestraCP(CP &valor)
{
    cout << valor.numCP << ";" << valor.localidad << ";" << endl;
    valor.listaPaq->mostrarLista();
}
```

```
void muestraEstadisticaCP(CP &valor)
{
    double frec_abs = valor.listaPaq->numNodos();
    double nCae = CAE->numNodos();
    double frec_rel = frec_abs / (100 - nCae);
    cout << setw(4) << valor.numCP << setw(12) << " " << setw(2) << frec_abs << setw(20) << " " << setw(2)
    << round(frec_rel*100) <<'%'<< endl;
}
```

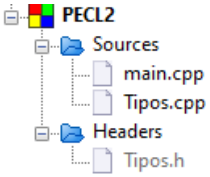
Estos son los métodos más importantes para definir las estructuras de datos utilizadas en el programa. A continuación se procederá a explicar su ejecución (en el siguiente apartado).

PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

Explicación del funcionamiento del programa.

Siguiendo la definición y especificación explicadas anteriormente de los TADs, se ha realizado la siguiente implementación:

Se han creado las siguientes clases implementando los ficheros:



En las cabeceras (Tipos.h) se ha definido la estructura de los datos, y en los ficheros origen (.cpp) se ha realizado la implantación de las funciones y/o métodos.

A continuación se realizará una explicación de su ejecución:

- Establecemos los valores por defecto.

```
int main()
{
    srand(time(NULL));
    int opcion;
    bool repetir = true;
    Paquete p;
    CP centralPaq;
    CP cpAux;
    Paquete paqueteAux;
    int i=0, j=0, nPaquetes=0, nCP=0;
    int paquetesCogidos=0;
    int codCentralCase2 = 0;
    int codCentralCase3 = 0;
    int codCentralCase6 = 0;
    int codCentralCase7 = 0;
    int codCentralCase8 = 0;

    string idCase5;
    string idCase6;
    string idCase7;
    string idCase8;
    //CP cpSelec;
    int gradLat=0, minLat=0, segLat=0, gradLong=0, minLong=0, segLong=0;
    int contadorPaquetes=N1;

    cout << string(33, '#') << "ALMACEN DE PAQUETES" << string(33, '#') << endl;
    cout << string(27, '=') << "LISTADO DE CENTRALES DE PAQUETERIA" << string(27, '=') << endl;
    cout << setw(4) << " No." << "|" << setw(7) << "ID CP " << "|" << setw(9) << "   Nombre localidad   " << endl;
    cout << string(45, '-') << endl;
```

```
##################################ALMACEN DE PAQUETES#################################
============================LISTADO DE CENTRALES DE PAQUETERIA============================
 No.| ID CP |   Nombre localidad
-------------------------------------------
  1     1848       Villarejo
  2     9849         Arganda
  3     1072         Arganda
  4     7834       Villarejo
  5     2237    Nuevo Baztan
  6     1209           Alcala
  7     2606    Camarma de Esteruelas
  8     2828        Mejorada
```

- Nos creamos el arbol de búsqueda y la lista de paquetes.

```
void inicializaSistema()
{
    // Inicializamos árbol de búsqueda y lista maestra de paquetes
    arbolCP = new Arbol();
    CAE = new Lista();
    int aux[N1];
    Paquete p;
```

PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

- Nos creamos las centrales de paquetería y las mostramos por pantalla.

```
for (int i=0; i<N1; i++)//Crea las 8 centrales de paqueteria y las muestra por pantalla
{
    CP cp=generarCpPredeterminado();
    aux[i]=cp.numCP;
    arbolCP->insertar(cp);
    cout<<setw(3)<<i+1<<" "<< setw(8) << cp.numCP << " " <<"   "<< setw(12) << cp.localidad << endl;//Ver como se hace con abb y no con array
}

cout << string(27, '=') << "LISTADO DE PAQUETES" << string(27,'=') << endl;
cout << setw(4) << " No." << "|" << setw(7) << "ID Paq " << "|" << setw(9) << "   NIF   " << "|" << setw(21) << "     Coordenadas     " << "|" << endl;
cout << string(22, ' ') << "|" << setw(10) << " Latitud  " << "|"<< setw(10) << " Longitud " << "|" << endl;
cout << string(45, '-') << endl;
```

- Nos creamos los N2 paquetes y los mostramos por pantalla.

```
//Crea los 100 paquetes y los muestra por pantalla
for (int j=0; j<N2; j++)
{
    // Seleccionamos aleatoriamente uno de los numCp generados en el paso anterior
    p=generarPaquete(aux[rand()%N1]);
    CAE->insertarNodo(p,'p');
    cout << setw(4)<< j+1 << " " << setw(7) << p.idPaquete << " " << setw(9) << p.NIF << " " << setw(2) << p.coordenadas.latitud[0]
    << "*" << setw(2) << p.coordenadas.latitud[1] << "'" << setw(2) << p.coordenadas.latitud[2] << "''" << " " << setw(2)
    << p.coordenadas.longitud[0] << "*"<< setw(2) << p.coordenadas.longitud[1] << "'" << setw(2) << p.coordenadas.longitud[2] << "''" << endl;
}
```

```
============================LISTADO DE PAQUETES============================
 No.|ID Paq |    NIF    |     Coordenadas     |
                        | Latitud  | Longitud |
-------------------------------------------
   1 02Z0306 40249587R 40*19'16'' -3* 7'34''
   2 18G8534 40601132Z 40*10'53'' -3*25' 1''
   3 76K3767 40553928Y 40*33' 0'' -3*10'31''
   4 90L1153 40135315Q 40* 6'28'' -3*18'38''
   5 79N2325 40971520X 40*27'14'' -3*11'31''
   6 36L7129 40035903X 40*16'46'' -3*19'40''
   7 45W4676 40783474N 40*18'41'' -3*18'31''
   8 70Q5129 40818285R 40* 6' 2'' -3*24'17''
   9 35M4054 40455347A 40*12'50'' -3*26' 1''
  10 27Z0028 40231606Y 40*10' 1'' -3*26'37''
  11 84Z1252 40417329G 40*20' 2'' -3*12'18''
  12 47K0189 40909148Z 40* 7'49'' -3*27'56''
  13 21G4691 40184741S 40*27'10'' -3*17'17''
  14 48M7771 40502902V 40*31' 1'' -3* 5'40''
  15 95R3602 40077540V 40*30' 0'' -3*30'36''
```

- Trasladamos los paquetes como se indica en el enunciado de la práctica.

```
// Bucle de iteraciones/días
int contador = 0;
while(contador++<N5)
{
    sigInstruccion();
    paquetesCogidos=0;
    cout << "Paquetes enviados en el dia " << contador << ":"<<endl;
    while(paquetesCogidos++ < N3)
    {
        pNodoLista nodoLista = CAE->borrarAntiguo();
        if (nodoLista!=NULL)
        {
            p = nodoLista->getPaquete();
            CP cpAux;
            cpAux.numCP = p.numCP;
            pNodoArbol nodoArbol = arbolCP->buscar(cpAux);
            nodoArbol->getCentrarlP().listaPaq->insertarNodo(p,'p');
            delete(nodoLista);
            cout<< "Paquete: "<<p.idPaquete << " añadido a la lista de paquetes del CP de :" <<nodoArbol->getCentrarlP().numCP << endl;
        }
    }
}
```

```
Paquetes enviados en el dia 1:
Paquete: 02Z0306 a▒adido a la lista de paquetes del CP de :2606
Paquete: 18G8534 a▒adido a la lista de paquetes del CP de :2237
Paquete: 76K3767 a▒adido a la lista de paquetes del CP de :9849
Paquete: 90L1153 a▒adido a la lista de paquetes del CP de :1072
Paquete: 79N2325 a▒adido a la lista de paquetes del CP de :1072
Paquete: 36L7129 a▒adido a la lista de paquetes del CP de :2237
Paquete: 45W4676 a▒adido a la lista de paquetes del CP de :2828
Paquete: 70Q5129 a▒adido a la lista de paquetes del CP de :1848
Paquete: 35M4054 a▒adido a la lista de paquetes del CP de :2828
Paquete: 27Z0028 a▒adido a la lista de paquetes del CP de :1848
Paquete: 84Z1252 a▒adido a la lista de paquetes del CP de :7834
Paquete: 47K0189 a▒adido a la lista de paquetes del CP de :9849

Presiona Enter para realizar la siguiente instruccion...
```

- Presentamos el menú de opciones.

```
do
{
    system("cls");
    // Texto del menú que se verá cada vez
    cout << "\n\nMenu de Opciones" << endl;
    cout << "1. Insertar una CP de forma manual." << endl;
    cout << "2. Borrar una CP del arbol." << endl;
    cout << "3. Mostrar los datos de los paquetes que se distribuiran en una CP dada." << endl;
    cout << "4. Mostrar una estadistica de las CP de la empresa recorriendo Postorden los nodos del arbol." << endl;
    cout << "5. Buscar un paquete concreto por su ID." << endl;
    cout << "6. Extraer algun paquete concreto de una CP dada (borrarlo del sistema)." << endl;
    cout << "7. Llevar un paquete concreto del CAE a una CP concreta." << endl;
    cout << "8. Llevar un paquete concreto de una CP a otra." << endl;
    cout << "9. Continuar con la distribucion de paquetes." << endl;

    cout << "0. Salir del programa" << endl;

    cout << "\nIngrese una opcion: ";
    cin >> opcion;
```

```
Menu de Opciones
1. Insertar una CP de forma manual.
2. Borrar una CP del arbol.
3. Mostrar los datos de los paquetes que se distribuiran en una CP dada.
4. Mostrar una estadistica de las CP de la empresa recorriendo Postorden los nodos del arbol.
5. Buscar un paquete concreto por su ID.
6. Extraer algun paquete concreto de una CP dada (borrarlo del sistema).
7. Llevar un paquete concreto del CAE a una CP concreta.
8. Llevar un paquete concreto de una CP a otra.
9. Continuar con la distribucion de paquetes.
0. Salir del programa

Ingrese una opcion:
```

PECL 2 - Estructuras de Datos. Simulación del funcionamiento de una red comarcal de paquetería.

A continuación se procederá a explicar las diferentes opciones:

1. Insertar una CP de forma manual.

```
case 1:
{
    // Insertar una CP de forma manual
    cout<<"Numero de CP deseado(Numero de tres cifras): ";
    cin >>centralPaq.numCP;//Asigno el número de CP manual
    for(auto const& value : localidades) //Para ver las localidades disponibles.
    {
        cout << value << ", ";
    }
    cout << endl;
    cout << "Localidad deseada: ";
    cin >> centralPaq.localidad;
    cout << endl;
    centralPaq.listaPaq = new Lista();
    arbolCP->insertar(centralPaq);
    arbolCP->inOrden(&muestraNodoArbol);
    cout << "¿Quieres continuar con la distribucion de paquetes? (Si/No): ";
    string r;
    cin >> r;
    if (r == "Si")
    {
        for(int i=0; i<12; i++)
        {
            p=generarPaquete(centralPaq.numCP);
            centralPaq.listaPaq->insertarNodo(p,'p');
        }
        arbolCP->inOrden(&muestraNodoArbol);
    }
    else
    {
        arbolCP->inOrden(&muestraNodoArbol);

    }
    system("pause>nul"); // Pausa
    break;
```

```
Ingrese una opcion:
1
Numero de CP deseado(Numero de tres cifras): 123
Ajalvir, Daganzo, Alcala, Mejorada, Nuevo Baztan, Arganda, Carabanna, Chinchon, Villarejo, Camarma de Esteruelas, Meco, Cobe
a, Torres de la Alameda, Los Santos de la Humosa,
Localidad deseada: Ajalvir

123;
767;
1072;
1209;
1848;
2237;
2606;
2828;
7834;
9849;
Quieres continuar con la distribucion de paquetes? (Si/No):
```

2. Borrar una CP del arbol.

```
case 2:
    // Borrar una CP del árbol
    cout << "Lista de centrales disponibles:"<<endl;
    arbolCP->inOrden(&muestraNodoArbol);
    cout << "Codigo de la central que desea eliminar: "; //Busco la central por su código
    cin >> cpAux.numCP;
    //cpAux.numCP=codCentralCase2;
    cout << endl;
    arbolCP->borrar(cpAux);

    arbolCP->inOrdenCP(&muestraCP);
    system("pause>nul"); // Pausa
    break;
```

```
Ingrese una opcion: 2
Lista de centrales disponibles:
123;
767;
1072;
1209;
1848;
2237;
2606;
2828;
7834;
9849;
Codigo de la central que desea eliminar: 123
```

3. Mostrar los datos de los paquetes que se distribuiran en una CP dada.

```
case 3:
    // Mostrar los datos de los paquetes que se distribuirán en una CP dada
    arbolCP->inOrden(&muestraNodoArbol);

    cout << "Central que desea ver (Numero de tres cifras): ";
    cin >> codCentralCase3;
    cpAux.numCP=codCentralCase3;
    pNodoArbol auxNodoArbol;
    auxNodoArbol=arbolCP->buscar(cpAux);
    if (auxNodoArbol!=NULL)
    {
        auxNodoArbol->getCentrarlP().listaPaq->mostrarLista();
    }
    else
    {
        cout<<"No hay ninguna central con ese codigo"<<endl;
    }
    system("pause>nul"); // Pausa
    break;
```

```
Ingrese una opcion: 3
1169;
1480;
2019;
2536;
2695;
3587;
5745;
5790;
Central que desea ver (Numero de tres cifras): 2536
   1 05P6290 40235486E 40*16'16'' -3*17' 9''
   2 51J2933 40594317F 40*17' 8'' -3*28' 5''
   3 31N2189 40059398E 40*33'28'' -3*26'59''
   4 61M9118 40723369Y 40*26'40'' -3*26'38''
   5 26L9801 40794047M 40*14'32'' -3*23' 8''
   6 57F2178 40960751M 40*31'49'' -3*14'38''
   7 83W6569 40422397N 40* 9'26'' -3* 5'48''
   8 80S0924 40313452H 40*20'54'' -3* 8'47''
   9 20U1880 40341518R 40*29'14'' -3*30'51''
  10 82A2921 40408781N 40*11'39'' -3*17'27''
```

4. Mostrar una estadistica de las CP de la empresa recorriendo Postorden los nodos del arbol.

```
case 4:
    // Mostrar una estadistica de las CP de la empresa recorriendo Inorden los nodos del Arbol
    cout << setw(4) << "CP " << "|" << setw(7) << " Frecuencias absolutas " << "|" << setw(9) << " Frecuencias relativas" << endl;
    arbolCP->postOrdenCP(&muestraEstadisticaCP);
    system("pause>nul"); // Pausa
    break;
```

```
Ingrese una opcion: 4
 CP | Frecuencias absolutas | Frecuencias relativas
1169              10                    14%
2019               8                    11%
2536              10                    14%
1480              11                    15%
5745              11                    15%
5790               9                    13%
3587               8                    11%
2695               5                     7%
```

5. Buscar un paquete concreto por su ID.

```
case 5:
    // Buscar un paquete concreto por su ID
    arbolCP->inOrdenCP(&muestraCP);
    cout << "ID del paquete a buscar: ";
    cin >> idCase5;
    cout << endl;
    {
        Paquete p;
        pNodoArbol n;
        n = arbolCP->buscarNodoInOrdenCP(&encuentraNodo,idCase5);
        if (n==NULL)
        {
            cout << "No se ha encontrado el paquete " << idCase5 << endl;
        }
        else
        {
            CP cp = n->getCentrarlP();
            pNodoLista pn = cp.listaPaq->existe(idCase5);
            p = pn->getPaquete();
            cout << "El paquete encontrado es: " << setw(7) << p.idPaquete << " " << setw(9) << p.NIF << " " << setw(2)
                 << p.coordenadas.latitud[0] << "*" << setw(2) << p.coordenadas.latitud[1] << "'" << setw(2) << p.coordenadas.latitud[2]
                 << "''" << " " << setw(2) << p.coordenadas.longitud[0] << "*"<< setw(2) << p.coordenadas.longitud[1] << "'" << setw(2)
                 << p.coordenadas.longitud[2] << "''" << endl;
        }
    }
    system("pause>nul"); // Pausa
    break;
```

```
ID del paquete a buscar: 51H9300

encontrado
El paquete encontrado es: 51H9300 40099632Y 40*29'27'' -3* 9' 9''
```

6. Extraer algun paquete concreto de una CP dada (borrarlo del sistema).

```
case 6:
    // Extraer algún paquete concreto de una CP dada (borrarlo del sistema)
    arbolCP->inOrden(&muestraNodoArbol);
    cout << "Codigo de la central donde esta el paquete que quiere buscar: "; //Busco la central por su código
    cin >> codCentralCase6;
    cout << endl;
    cpAux.numCP=codCentralCase6;
    auxNodoArbol=arbolCP->buscar(cpAux);
      muestraCP(auxNodoArbol->getCentrarlP());
    cout << auxNodoArbol->getCentrarlP().numCP << ";" << auxNodoArbol->getCentrarlP().localidad << ";" << endl;
    auxNodoArbol->getCentrarlP().listaPaq->mostrarLista();

    cout << "ID del paquete a buscar: ";
    cin >> idCase6;
    cout << endl;
    pNodoArbol n;
    n = arbolCP->buscarNodoInOrdenCP(&encuentraNodo,idCase6);
    if (n==NULL)
    {
        cout << "No se ha encontrado el paquete " << idCase6 << endl;
    }
    else
    {
        CP cp = n->getCentrarlP();
        pNodoLista pn = cp.listaPaq->existe(idCase6);
                    Paquete p = pn->getPaquete();
        cout << "El paquete encontrado es: " << setw(7) << p.idPaquete << " " << setw(9) << p.NIF << " " << setw(2) <<
             p.coordenadas.latitud[0] << "*" << setw(2) << p.coordenadas.latitud[1] << "'" << setw(2) << p.coordenadas.latitud[2]
             << "''" << " " << setw(2) << p.coordenadas.longitud[0] << "*"<< setw(2) << p.coordenadas.longitud[1] << "'" << setw(2)
             << p.coordenadas.longitud[2] << "''" << endl;
        cp.listaPaq->borrarNodoIntermedio(pn);
        cout << "Paquete borrado" << endl;
    }
    system("pause>nul"); // Pausa
    break;
```

```
Ingrese una opcion: 6
1169;
1480;
2019;
2536;
2695;
3587;
5745;
5790;
Codigo de la central donde esta el paquete que quiere buscar: 2695

2695;Alcala;
   1 27G2581 40247023J 40*12'48'' -3*33'15''
   2 51H9300 40099632Y 40*29'27'' -3* 9' 9''
   3 00B0767 40726791R 40*10'49'' -3* 7'11''
   4 17E4620 40657122E 40*19'36'' -3*17'22''
   5 16A1178 40722586M 40*11' 1'' -3*32'59''
   6 28I8153 40792391M 40*32'34'' -3*27'37''
   7 36L5939 40970122S 40* 7'55'' -3*29'56''
ID del paquete a buscar: 00B0767

encontrado
El paquete encontrado es: 00B0767 40726791R 40*10'49'' -3* 7'11''
Paquete borrado
```

7. Llevar un paquete concreto del CAE a una CP concreta.

```
case 7:
    // Llevar un paquete concreto del CAE a una CP concreta
    CAE->mostrarLista();
    cout << "ID del paquete a enviar: ";
    cin >> idCase7;
    cout << endl;
    arbolCP->inOrdenCP(&muestraCP);
    cout << "Codigo de la central de destino: "; //Busco la central por su código
    cin >> codCentralCase7;
    cout << endl;
    {
        pNodoLista n = CAE->existe(idCase7);
        if (n!=NULL)
        {
            Paquete p = n->getPaquete();
            CP cp;
            cp.numCP=codCentralCase7;
            pNodoArbol pna = arbolCP->buscar(cp);
            pna->getCentrarlP().listaPaq->insertarNodo(p,'p');
        }
        cout << "Paquete remitido" << endl << endl;
        cout << "Lista de CPs y paquetes actualizada: " << endl;

        arbolCP->inOrdenCP(&muestraCP);
    }
    system("pause>nul"); // Pausa
    break;
```

```
Ingrese una opcion: 7
   1 86C8910 40298581M 40* 8'54'' -3*24'51''
   2 09A4884 40562783Y 40*14'13'' -3*30'23''
   3 59E5851 40381303L 40* 7'43'' -3* 9'52''
   4 08D8360 40823841Z 40*23' 8'' -3*31'34''
   5 53J8876 40581117D 40*21'16'' -3* 8'43''
   6 28H1162 40187222N 40*13'33'' -3*15'38''
   7 23W4180 40183536Y 40*18'23'' -3*22'51''
   8 11H9928 40762398G 40*11'49'' -3* 8'19''
   9 70M3548 40980815J 40*29' 8'' -3* 5' 9''
  10 22Q3914 40945754G 40*21'55'' -3*25'19''
  11 31J0034 40684664X 40*28'36'' -3*20'36''
  12 46I9515 40606074B 40*21'10'' -3*15'24''
  13 75W0288 40920590w 40*15'31'' -3*17'27''
  14 64S5710 40212344H 40*10'37'' -3*15'53''
  15 86X1778 40474399B 40* 6'24'' -3*23'44''
  16 83B2238 40561826S 40*16'29'' -3*32'36''
ID del paquete a enviar: 64S5710
```

```
5790;Camarma de Esteruelas;
   1 39G2921 40842704V 40* 9'24'' -3*15'59''
   2 93M6709 40189276L 40*24' 5'' -3* 5'58''
   3 32U5072 40314747w 40*33'12'' -3*17'33''
   4 93G0824 40812115H 40*25'13'' -3*23'15''
   5 43S3231 40760240P 40*24'11'' -3*30'10''
   6 26S0829 40736018M 40*10'24'' -3* 6'32''
   7 05R0737 40263783Y 40* 8'52'' -3*13'12''
   8 62X1222 40014727V 40*18'28'' -3* 8'35''
   9 26U6662 40037020T 40*11'47'' -3*32'20''
  10 65X2303 40664722D 40*25'33'' -3*15'14''
  11 13B0392 40618872K 40*14'26'' -3*13'36''
Codigo de la central de destino: 5790
```

```
5790;Camarma de Esteruelas;
   1 64S5710 40212344H 40*10'37'' -3*15'53''
   2 39G2921 40842704V 40* 9'24'' -3*15'59''
   3 93M6709 40189276L 40*24' 5'' -3* 5'58''
   4 32U5072 40314747w 40*33'12'' -3*17'33''
   5 93G0824 40812115H 40*25'13'' -3*23'15''
   6 43S3231 40760240P 40*24'11'' -3*30'10''
   7 26S0829 40736018M 40*10'24'' -3* 6'32''
   8 05R0737 40263783Y 40* 8'52'' -3*13'12''
   9 62X1222 40014727V 40*18'28'' -3* 8'35''
  10 26U6662 40037020T 40*11'47'' -3*32'20''
  11 65X2303 40664722D 40*25'33'' -3*15'14''
  12 13B0392 40618872K 40*14'26'' -3*13'36''
```

8. Llevar un paquete concreto de una CP a otra.

```
case 8:
{

    // Llevar un paquete concreto de una CP a otra
    // Extraer algún paquete concreto de una CP dada (borrarlo del sistema)
    arbolCP->inOrden(&muestraNodoArbol);
    cout << "Codigo de la central donde esta el paquete que quiere buscar: "; //Busco la central por su código
    cin >> codCentralCase8;
    cout << endl;
    cpAux.numCP=codCentralCase8;
    auxNodoArbol=arbolCP->buscar(cpAux);
      muestraCP(auxNodoArbol->getCentrarlP());
    cout << auxNodoArbol->getCentrarlP().numCP << ";" << auxNodoArbol->getCentrarlP().localidad << ";" << endl;
    auxNodoArbol->getCentrarlP().listaPaq->mostrarLista();

    cout << "ID del paquete a buscar: ";
    cin >> idCase8;
    cout << endl;
    n = arbolCP->buscarNodoInOrdenCP(&encuentraNodo,idCase8);
    if (n==NULL)
    {
        cout << "No se ha encontrado el paquete " << idCase8 << endl;
    }
    else
    {
        CP cp = n->getCentrarlP();
        pNodoLista pn = cp.listaPaq->existe(idCase8);
        Paquete p = pn->getPaquete();
        cout << "El paquete encontrado es: " << setw(7) << p.idPaquete << " " << setw(9) <<
        p.NIF << " " << setw(2) << p.coordenadas.latitud[0] << "" << setw(2) << p.coordenadas.latitud[1]
        << "'" << setw(2) << p.coordenadas.latitud[2] << "'" << " " << setw(2) << p.coordenadas.longitud[0]
        << ""<< setw(2) << p.coordenadas.longitud[1] << "'" << setw(2) << p.coordenadas.longitud[2] << "'" << endl;
        cp.listaPaq->borrarNodoIntermedio(pn);
        cout << "Paquete borrado" << endl;
    }

        arbolCP->inOrdenCP(&muestraCP);
        cout << "Central que desea ver: ";
        cin >> codCentralCase8;
        CP cpa;
        cpa.numCP=codCentralCase8;
        pNodoArbol auxn;
        auxn=arbolCP->buscar(cpa);
        auxn->getCentrarlP().listaPaq->insertarNodo(p,'p');




        cout << "Paquete remitido" << endl << endl;
        cout << "Lista de CPs y paquetes actualizada: " << endl;
        arbolCP->inOrdenCP(&muestraCP);

        system("pause>nul"); // Pausa
        break;
}
```

```
Ingrese una opcion: 8
1169;
1480;
2019;
2536;
2695;
3587;
5745;
5790;
Codigo de la central donde esta el paquete que quiere buscar: 5745
```

```
5745;Carabanna;
   1 59R2606 40060261B 40*22'57'' -3* 8'50''
   2 20C2262 40082552S 40*29'36'' -3* 5'59''
   3 70I0209 40008310V 40*19'46'' -3*28'15''
   4 13D6797 40124050K 40*14' 1'' -3*19'45''
   5 13R0792 40628283w 40*26' 1'' -3*15'21''
   6 85S5798 40992070K 40*18'17'' -3*17'28''
   7 18G7635 40318581H 40*13'26'' -3* 8' 3''
   8 65I1312 40930161M 40*12'23'' -3*33'36''
   9 01P8134 40204447X 40*24'54'' -3*33'22''
  10 92F0387 40656989G 40*17'50'' -3*21'17''
  11 85E4531 40881638N 40*20'56'' -3*25'54''
ID del paquete a buscar: 01P8134
```

```
Central que desea ver: 5790
Paquete remitido
```

```
5790;Camarma de Esteruelas;
   1 39G2921 40842704V 40* 9'24'' -3*15'59''
   2 64S5710 40212344H 40*10'37'' -3*15'53''
   3 39G2921 40842704V 40* 9'24'' -3*15'59''
   4 93M6709 40189276L 40*24' 5'' -3* 5'58''
   5 32U5072 40314747w 40*33'12'' -3*17'33''
   6 93G0824 40812115H 40*25'13'' -3*23'15''
   7 43S3231 40760240P 40*24'11'' -3*30'10''
   8 26S0829 40736018M 40*10'24'' -3* 6'32''
   9 05R0737 40263783Y 40* 8'52'' -3*13'12''
  10 62X1222 40014727V 40*18'28'' -3* 8'35''
  11 26U6662 40037020T 40*11'47'' -3*32'20''
  12 65X2303 40664722D 40*25'33'' -3*15'14''
  13 13B0392 40618872K 40*14'26'' -3*13'36''
```

9. Continuar con la distribucion de paquetes.

```
// Continuar con la distribución de paquetes
bool seguir = true;
while(!CAE->listaVacia()&& seguir)
{
    sigInstruccion();
    paquetesCogidos=0;
    cout << "Paquetes enviados en el dia " << contador << ":"<<endl;
    while(paquetesCogidos++ < N3)
    {
        pNodoLista nodoLista = CAE->borrarAntiguo();
        if (nodoLista!=NULL)
        {
            p = nodoLista->getPaquete();
            CP cpAux;
            cpAux.numCP = p.numCP;
            pNodoArbol nodoArbol = arbolCP->buscar(cpAux);
            nodoArbol->getCentrarlP().listaPaq->insertarNodo(p,'p');
            delete(nodoLista);
            cout<< "Paquete: "<<p.idPaquete << " annadido a la lista de paquetes del CP de :" <<nodoArbol->getCentrarlP().numCP << endl;
        }
    }
    cout << "¿Quieres continuar con la distribucion de paquetes? (Si/No): ";
    string r;
    cin >> r;
    if (r == "Si")
    {
        contador++;
    }
    else
    {
        contador++;
        seguir = false;
    }
}
system("pause>nul"); // Pausa
break;
```

```
Ingrese una opcion: 9

Presiona Enter para realizar la siguiente instruccion...

Paquetes enviados en el dia 7:
Paquete: 51H9300 annadido a la lista de paquetes del CP de :2695
Paquete: 23Y2230 annadido a la lista de paquetes del CP de :2536
Paquete: 93M6709 annadido a la lista de paquetes del CP de :5790
Paquete: 09T3505 annadido a la lista de paquetes del CP de :3587
Paquete: 74G1343 annadido a la lista de paquetes del CP de :2536
Paquete: 19Z2961 annadido a la lista de paquetes del CP de :2019
Paquete: 15J0174 annadido a la lista de paquetes del CP de :1480
Paquete: 14P6557 annadido a la lista de paquetes del CP de :1480
Paquete: 39C5498 annadido a la lista de paquetes del CP de :1169
Paquete: 27G2581 annadido a la lista de paquetes del CP de :2695
Paquete: 96W1822 annadido a la lista de paquetes del CP de :1480
Paquete: 39G2921 annadido a la lista de paquetes del CP de :5790
¿Quieres continuar con la distribucion de paquetes? (Si/No):
```

Problemas encontrados durante el desarrollo de la práctica y solución adoptada.

Creación del arbol binario:

- Se ha optado por la especificacion explicada en clase creando los nodos de la siguiente forma:

```
class NodoArbol {
    private:
        // Miembros:
        CP centralP;
        NodoArbol *izquierdo;
        NodoArbol *derecho;
        friend class Arbol;

    public:
        // Constructor:
        NodoArbol(CP cp, NodoArbol *izq=NULL, NodoArbol *der=NULL) :
        centralP(cp), izquierdo(izq), derecho(der) {};
        CP getCentrarlP(){
            return centralP;
        };

};
typedef NodoArbol *pNodoArbol;
```

- Y la especificación de la clase de esta manera:

```
class Arbol {
    private:
        // Punteros de la lista, para cabeza y nodo actual:
        pNodoArbol raiz;
        pNodoArbol actual;
        int contador;
        int altura;

    public:
        // Constructor y destructor básicos:
        Arbol() : raiz(NULL), actual(NULL) {}
        ~Arbol() { podar(raiz); }
        // Insertar en árbol ordenado:
        void insertar(CP cp);
        // Borrar un elemento del árbol:
        void borrar(CP cp);
        // Función de búsqueda:
        pNodoArbol buscar(CP cp);
        // Comprobar si el árbol está vacío:
        bool vacio(pNodoArbol r) { return r==NULL; }
        // Comprobar si es un nodo hoja:
        bool esHoja(pNodoArbol r) { return r->derecho!=NULL && r->izquierdo!=NULL; }
        // Contar número de nodos:
        const int numeroNodos();
        const int alturaArbol();
        // Calcular altura de un int:
        int calculaAltura(CP cp);
        // Devolver referencia al int del nodo actual:
        CP cpActual() { return actual->centralP; } //&ValorActual()
        // Moverse al nodo raiz:
        void volverARaiz() { actual = raiz; }
        // Aplicar una función a cada elemento del árbol:
        void inOrden(void (*func)(int&) , pNodoArbol nodo=NULL, bool r=true);
        void preOrden(void (*func)(int&) , pNodoArbol nodo=NULL, bool r=true);
        void postOrden(void (*func)(int&) , pNodoArbol nodo=NULL, bool r=true);
        void inOrdenCP(void (*func)(CP&) , pNodoArbol nodo=NULL, bool r=true);
        void preOrdenCP(void (*func)(CP&) , pNodoArbol nodo=NULL, bool r=true);
        void postOrdenCP(void (*func)(CP&) , pNodoArbol nodo=NULL, bool r=true);
        Paquete *buscarInOrdenCP(Paquete *(*func)(CP&, string),string idPaquete, pNodoArbol nodo=NULL, bool r=true);
        pNodoArbol buscarNodoInOrdenCP(bool (*func)(CP&, string),string idPaquete, pNodoArbol nodo=NULL, bool r=true);
        private:
        // Funciones auxiliares
        void podar(pNodoArbol &nodo);
        void auxContador(pNodoArbol nodo);
        void auxAltura(pNodoArbol nodo, int alt);
};
```

Errores varios:

- Se han encontrado problemas al intentar cambiar la posición de los nodos del arbol binario.
- Se han encontrado problemas al añadir nuevos nodos y eliminarlos en el arbol binario sin corromper los datos del arbol.

Los problemas anteriores se han solucionado como se ha explicado en el apartado de la explicación de los métodos.