Documentación

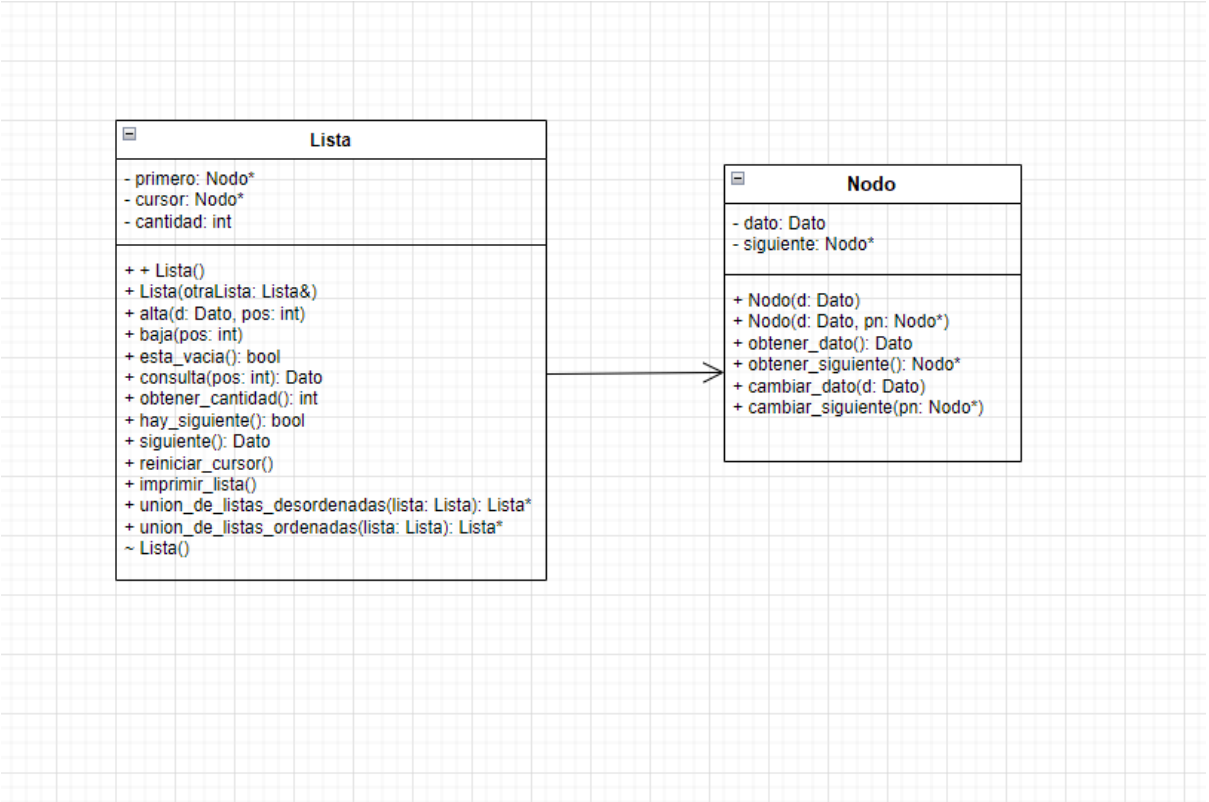
Ej 1) Lista:

```
private:
        // pre: 0 < pos <= cantidad
        // pos: devuelve un puntero al nodo en la posicion pos
        Nodo* obtener_nodo(int pos);
public:
   // CONSTRUCTOR
   // pre: -
   // pos: construye una Lista
   Lista();

   // CONSTRUCTOR DE COPIA
   // pre: -
   // pos:
   Lista(Lista& otraLista);

   // METODOS PUBLICOS

   // pre: 0 < pos <= obtener_cantidad() + 1
   // pos: agrega d a la Lista en la posicion pos
   void alta(Dato d, int pos);

   // pre: 0 < pos <= obtener_cantidad()
   // pos: elimina de la Lista el elemento en la posicion pos
   void baja(int pos);

   // pre: -
   // pos: devuelve true si la Lista esta vacia
   // devuelve false en caso contrario
   bool esta_vacia();

   // pre: 0 < pos <= obtener_cantidad()
   // pos: devuelve el elemento de la posicion pos
   Dato consulta(int pos);

   // pre: -
   // pos: devuelve la cantidad de elementos en la lista
   int obtener_cantidad();

   // pre: -
   // pos: devuelve true si hay siguiente, false si no
   bool hay_siguiente();
```

```
    // pre: hay_siguiente() == true
    // pos: devuelve el siguiente dato y actualiza el cursor
    Dato siguiente();

    // pre: -
    // pos: vuelve el cursor al inicio
    void reiniciar_cursor();

    //pre:
    //pos: muestra por pantalla los elementos de la lista
    void imprimir_lista();

    //pre: la lista debe estar desordenada
    //pos: devuelve una lista como resultado de la union de las listas
    Lista* union_de_listas_desordenadas(Lista lista);

    //pre: la lista debe estar ordenada de menor a mayor
    //pos: devuelve una lista como resultado de la union de las listas ordenadas
    Lista* union_de_listas_ordenadas(Lista lista);

    // DESTRUCTOR
    // pre: -
    // pos: destruye la Lista y libera toda la memoria utilizada.
    ~Lista();
```

Nodo:

```
public:
    // pos: construye un Nodo con un dato d y siguiente en nullptr.
    Nodo(Dato d);
    // pos: construye un Nodo con un dato d y siguiente en pn
    Nodo(Dato d, Nodo* pn);

    // METODOS
    //pos: devuelve el dato
    Dato obtener_dato();
    //pos:devuelve el siguiente
    Nodo* obtener_siguiente();
    //pos: cambia el dato actual por el dato ingresado
    void cambiar_dato(Dato d);
    //pos: cambia el elemento siguiente por el ingresado
    void cambiar_siguiente(Nodo* pn);
```



Ej 2) Lista:

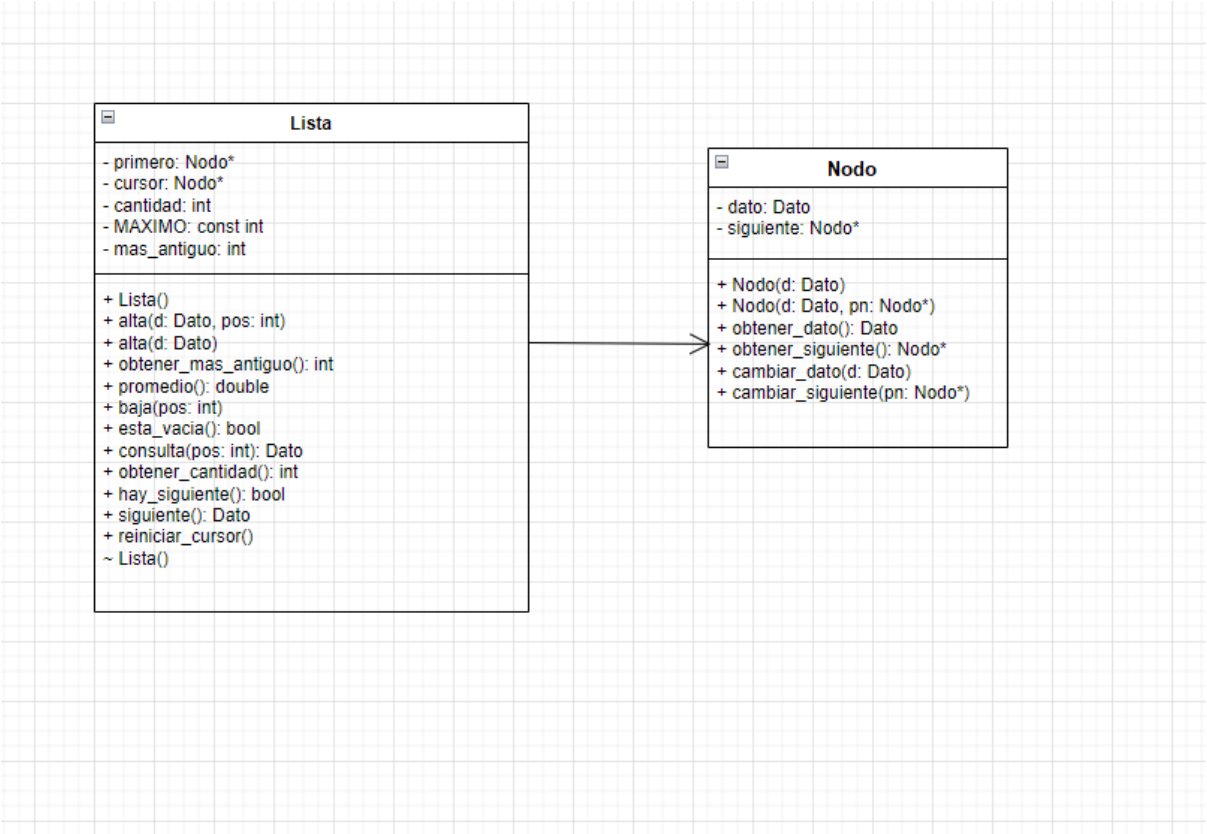
```
private:
// pre: 0 < pos <= cantidad
    // pos: devuelve un puntero al nodo en la posicion pos
    Nodo* obtener_nodo(int pos);
public:
// CONSTRUCTOR
    // pre: -
    // pos: construye una Lista
    Lista();

    // METODOS PUBLICOS

    // pre: 0 < pos <= obtener_cantidad() + 1
    // pos: agrega d a la Lista en la posicion pos
    void alta(Dato d, int pos);

    // pre:
    // pos: agrega d a la Lista
    void alta(Dato d);

    // pre:
    // pos: devuelve la posicion del mas antiguo
    int obtener_mas_antiguo();
```

```
// pre:
// pos: devuelve el promedio de los elementos de la lista
double promedio();

// pre: 0 < pos <= obtener_cantidad()
// pos: elimina de la Lista el elemento en la posicion pos
void baja(int pos);

// pre: -
// pos: devuelve true si la Lista esta vacia
// devuelve false en caso contrario
bool esta_vacia();

// pre: 0 < pos <= obtener_cantidad()
// pos: devuelve el elemento de la posicion pos
Dato consulta(int pos);

// pre: -
// pos: devuelve la cantidad de elementos en la lista
int obtener_cantidad();

// pre: -
// pos: devuelve true si hay siguiente, false si no
bool hay_siguiente();

// pre: hay_siguiente() == true
// pos: devuelve el siguiente dato y actualiza el cursor
Dato siguiente();

// pre: -
// pos: vuelve el cursor al inicio
void reiniciar_cursor();

// DESTRUCTOR
// pre: -
// pos: destruye la Lista y libera toda la memoria utilizada.
~Lista();
```

Nodo:

```
public:
// CONSTRUCTORES



    // pos: construye un Nodo con un dato d y siguiente en nullptr.
```

```
Nodo(Dato d);
// pos: construye un Nodo con un dato d y siguiente en pn
Nodo(Dato d, Nodo* pn);

// METODOS
//pos: devuelve el dato
Dato obtener_dato();
//pos:devuelve el siguiente
Nodo* obtener_siguiente();
//pos: cambia el dato actual por el dato ingresado
void cambiar_dato(Dato d);
//pos: cambia el elemento siguiente por el ingresado
void cambiar_siguiente(Nodo* pn);
```



Ej 3) Lista:

```
private:
// pre: 0 < pos <= cantidad
    // pos: devuelve un puntero al nodo en la posicion pos
    Nodo<Dato>* obtener_nodo(int pos);
public:
// CONSTRUCTOR
    // pre: -
    // pos: construye una Lista
```

```
Lista();

// METODOS PUBLICOS

// pre: 0 < pos <= obtener_cantidad() + 1
// pos: agrega d a la Lista en la posicion pos
void alta(Dato d, int pos);

// pre: 0 < pos <= obtener_cantidad()
// pos: elimina de la Lista el elemento en la posicion pos
void baja(int pos);

// pre: -
// pos: devuelve true si la Lista esta vacia
// devuelve false en caso contrario
bool esta_vacia();

// pre: 0 < pos <= obtener_cantidad()
// pos: devuelve el elemento de la posicion pos
Dato consulta(int pos);

// pre: -
// pos: devuelve la cantidad de elementos en la lista
int obtener_cantidad();

// pre: -
// pos: devuelve true si hay siguiente, false si no
bool hay_siguiente();

// pre: hay_siguiente() == true
// pos: devuelve el siguiente dato y actualiza el cursor
Dato siguiente();

// pre: -
// pos: vuelve el cursor al inicio
void reiniciar_cursor();

// DESTRUCTOR
// pre: -
// pos: destruye la Lista y libera toda la memoria utilizada.
~Lista();
```

Nodo:

```
public:
// CONSTRUCTORES
```

```
// pos: construye un Nodo con un dato d y siguiente en nullptr.
Nodo<Dato>(Dato d);
// pos: construye un Nodo con un dato d y siguiente en pn
Nodo<Dato>(Dato d, Nodo<Dato>* pn);

// METODOS
//pos: devuelve el dato
Dato obtener_dato();
//pos:devuelve el siguiente
Nodo* obtener_siguiente();
//pos: cambia el dato actual por el dato ingresado
void cambiar_dato(Dato d);
//pos: cambia el elemento siguiente por el ingresado
void cambiar_siguiente(Nodo* pn);
```

Carrera:

```
public:
/* pos: Crea una Carrera con su nombre, la duración (en años) y una lista
    de materias */
    Carrera (string nombre, int duracion, Lista<string>* materias);

    /*pos: devuelve el nombre de la Carrera */
    string obtener_nombre();

    /*pos: devuelve la duración */
    int obtener_duracion ();

    /*pos: devuelve un puntero a la lista de materias */
    Lista<string>* obtener_materias ();
```

Buscador_carreras:

```
public:
    // Constructor
    Buscador_Carreras();
    /*
    Pre: agregar las necesarias
    Post: busca en “carreras” aquellas que tienen tres o más materias de
    la lista “materias_predilectas” y una duración menor o igual que
    duracion_maxima.
    Devuelve un puntero a una lista con las carreras que cumplen con estas
    características.
    */
```

Lista<Carrera*>* sugerir_carreras(Lista<Carrera*>* carreras, int duracion_maxima, Lista<string>* materias_predilectas);

// pre: recibe por parametro un puntero a la lista resultante de la funcion sugerir_carreras
// pos: muestra por pantalla las carreras
void imprimir_carreras(Lista<Carrera*>* carreras_sugeridas);