Trabajo Practico n°3:

: *Curso de verano 2024 - Algoritmos y Programación II :*

Universidad de Buenos Aires Facultad de Ingenieria

Nombre: Natanael Daniel Brizuela.

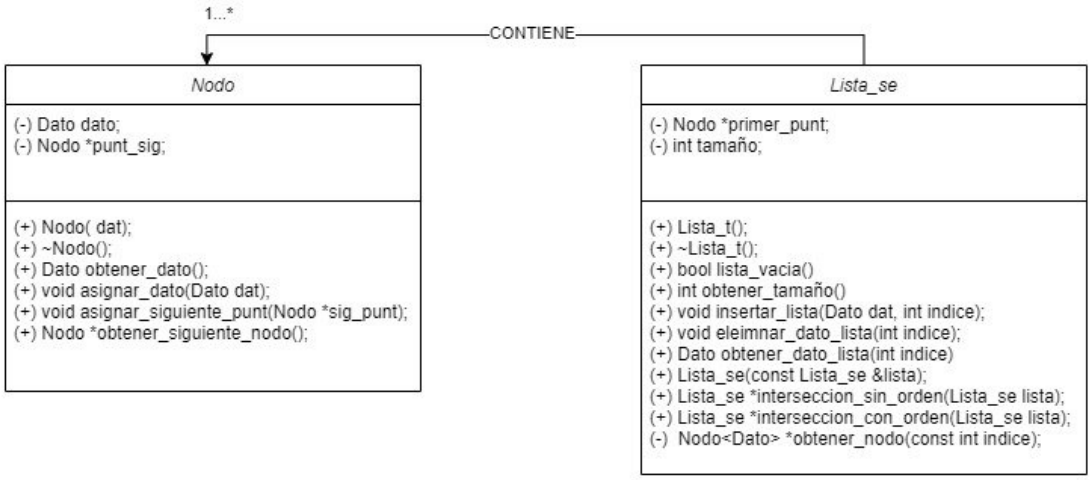
Padron: 110267

DOCUMENTACION

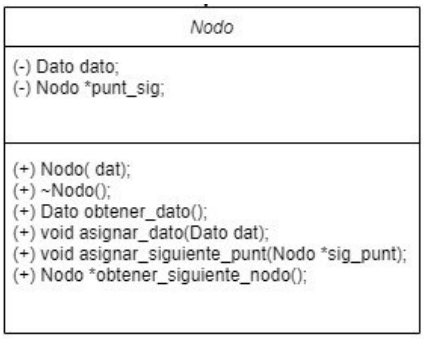
- o Diagrama de clases UML
- o Descripción de cada TDA, indicando las pre y poscondiciones de cada una de las operaciones.

Ejercicio 1

DIAGRAMA UML



:**Descripciones:**



Descripcion: La clase Nodo es una clase básica que representa un nodo en una estructura de datos enlazada. Un nodo contiene un dato y un puntero al siguiente nodo en la lista.

Atributos:

NODO

Dato dato: El dato almacenado en el nodo. Dato se puede definir como ‘char’, ‘int’, ‘string’ ...etc.

punt_sig: Un puntero al siguiente nodo en la lista.

Métodos:

Nodo(Dato dat): Constructor que crea un nuevo nodo con el dato especificado.

~Nodo(): Destructor que libera la memoria asignada al nodo.

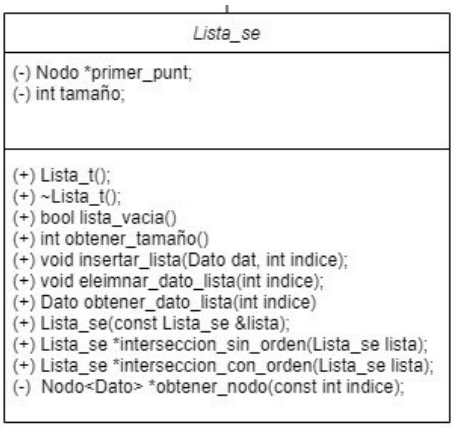
void asignar_dato(Dato dat): Modifica el dato almacenado en el nodo.

Dato obtener_dato(): Obtiene el dato almacenado en el nodo.

void asignar_siguiente_punt(Nodo *punt_sig): Modifica el puntero al siguiente nodo.

Nodo *obtener_siguiente_nodo(): Obtiene el puntero al siguiente nodo.

Descripcion: La clase Lista_se representa una lista enlazada simple. La lista contiene una colección de elementos de tipo Dato y cada elemento está enlazado al siguiente.



Atributos:

primero: Un puntero al primer nodo de la lista.

tamaño: La cantidad de elementos en la lista.

Métodos:

Constructores y destructores: **LISTA_SE**

Lista_se(): Constructor que crea una lista vacía.

~Lista_se(): Destructor que libera la memoria asignada a la lista.

Lista_se(const Lista_se &lista): Constructor de copia que crea una nueva lista con los mismos elementos que la lista original.

Seters:

void insertar_lista(Dato dat, int indice): Inserta un nuevo elemento con el dato especificado en la posición indicada por el índice.

void eleimnar_dato_lista(int indice): Elimina el elemento en la posición indicada por el índice.

Geters:

bool lista_vacia(): Devuelve true si la lista está vacía, false en caso contrario.

int obtener_tamaño(): Devuelve la cantidad de elementos en la lista.

Dato obtener_dato_lista(int indice): Devuelve el dato almacenado en el elemento en la posición indicada por el índice.

Operaciones con otras listas:

Lista_se *interseccion_sin_orden(Lista_se lista): Devuelve una nueva lista que contiene la intersección (elementos comunes) de las dos listas sin tener en cuenta el orden.

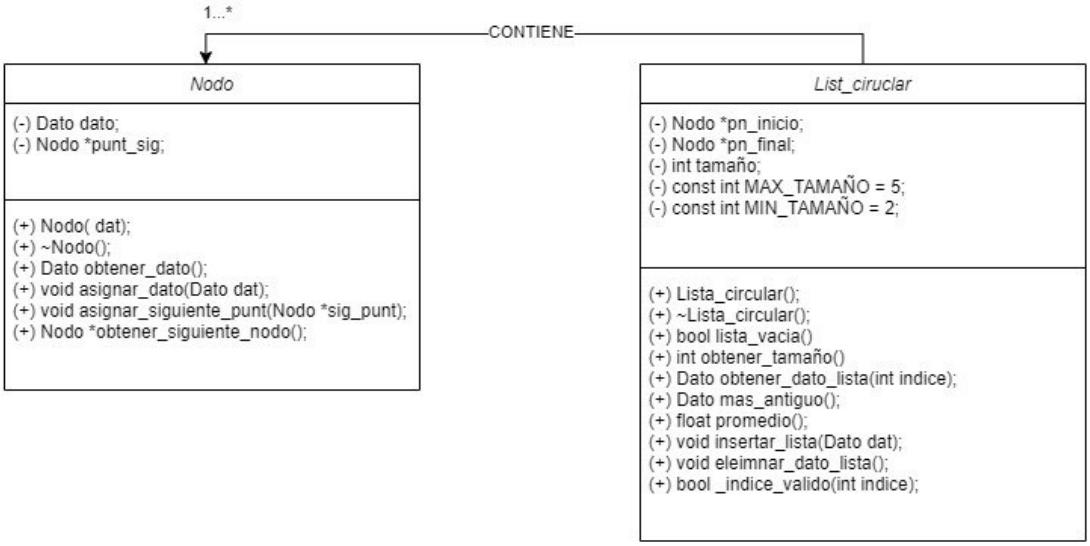
Lista_se *interseccion_con_orden(Lista_se lista): Devuelve una nueva lista que contiene la intersección (elementos comunes) de las dos listas ordenadas de mayor a menor.

Funciones auxiliares:

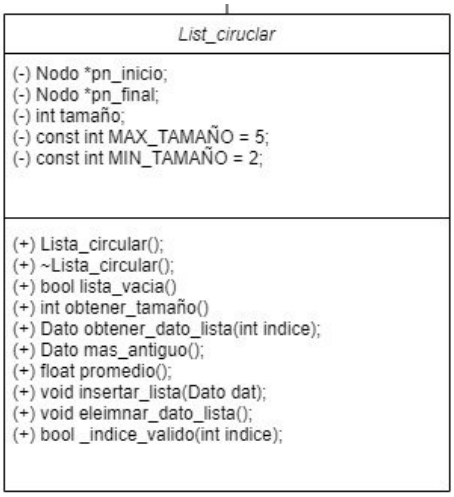
Nodo *obtener_nodo(int indice): Obtiene un puntero al nodo en la posición indicada por el índice.

Ejercicio 1

DIAGRAMA UML



Descripcion: La clase Lista_circular representa una lista circular enlazada. La lista contiene una colección de elementos de tipo Dato y cada elemento está enlazado al siguiente, formando un ciclo. El primer elemento y el último elemento de la lista están conectados.



LISTA_CIRULAR

Atributos:

pn_inicio: Un puntero al primer nodo de la lista.

pn_final: Un puntero al último nodo de la lista.

tamaño: La cantidad de elementos en la lista.

MAX_TAMAÑO: El tamaño máximo permitido de la lista.

MIN_TAMAÑO: El tamaño mínimo permitido de la lista.

Métodos:

Constructores y destructores:

Lista_circular(): Constructor que crea una lista vacía.

~Lista_circular(): Destructor que libera la memoria asignada a la lista.

Seters:

void insertar_lista(Dato dat): Inserta un nuevo elemento con el dato especificado al final de la lista.

void eleimnar_dato_lista(): Elimina el primer elemento de la lista.

Geters

bool lista_vacia(): Devuelve true si la lista está vacía, false en caso contrario.

unsigned obtener_tamaño(): Devuelve la cantidad de elementos en la lista.

Dato obtener_dato_lista(unsigned indice): Devuelve el dato almacenado en el elemento en la posición indicada por el índice.

Dato mas_antiguo(): Devuelve el dato del primer elemento de la lista.

float promedio(): Devuelve el promedio de los datos almacenados en la lista.

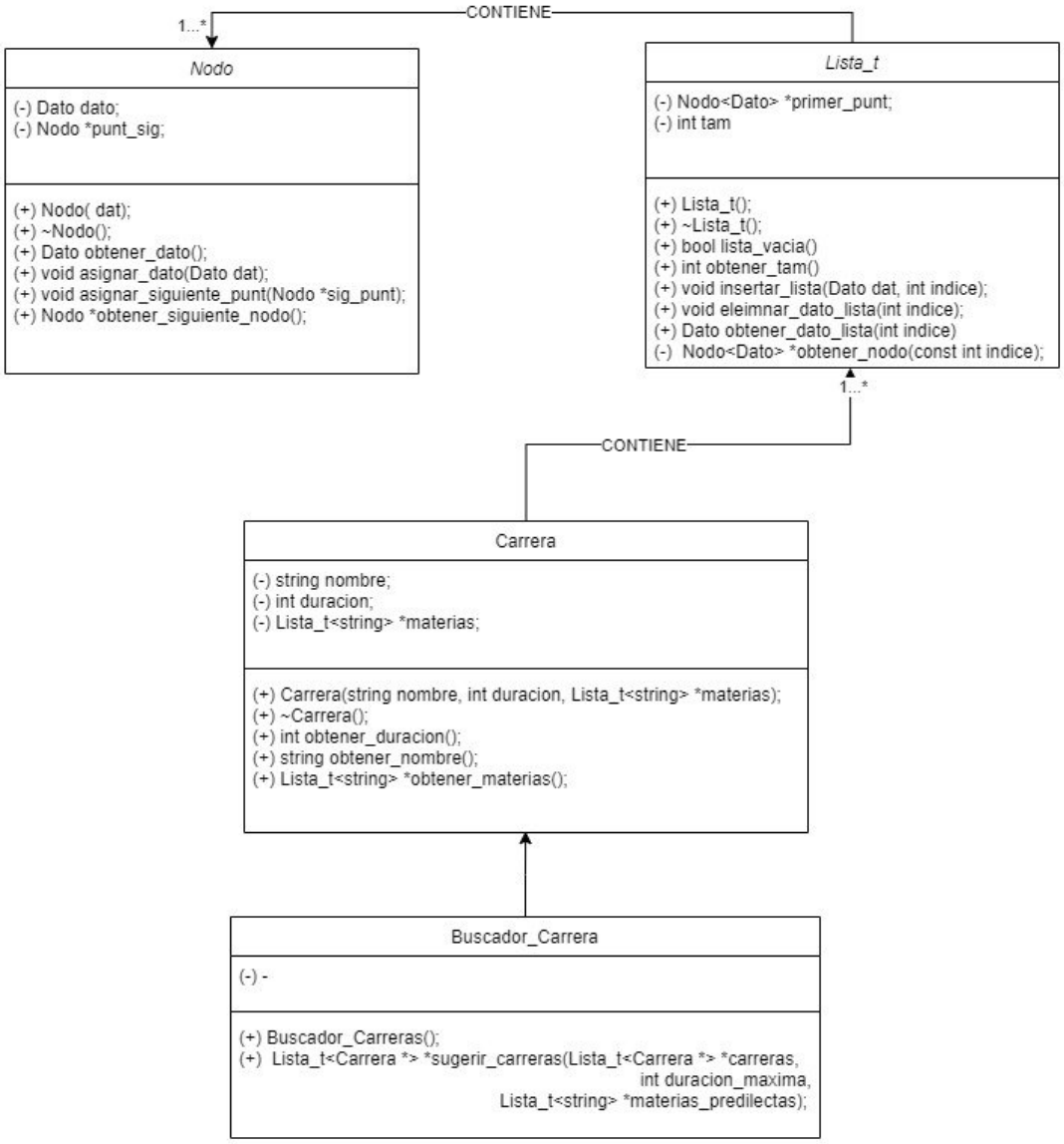
Funciones auxiliares:

Nodo *obtener_nodo(unsigned indice): Obtiene un puntero al nodo en la posición indicada por el índice.

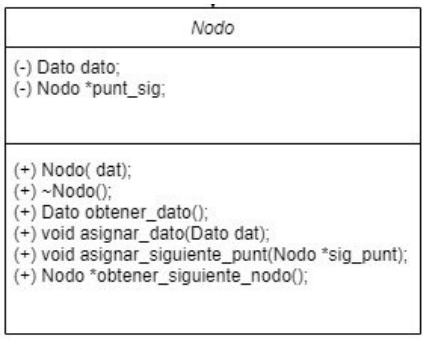
bool _indice_valido(unsigned indice): Valida si el índice está dentro del rango permitido.

Ejercicio 3

DIAGRAMA UML



Descripcion: La clase Nodo es una clase básica que representa un nodo en una estructura de datos enlazada. Un nodo contiene un dato y un puntero al siguiente nodo en la lista.



NODO (Template)

Atributos:

dato: El dato almacenado en el nodo.

punt_sig: Un puntero al siguiente nodo en la lista

Métodos:

Constructores y destructores:

Nodo(Dato dat): Constructor que crea un nuevo nodo con el dato especificado.

~Nodo(): Destructor que libera la memoria asignada al nodo.

Seters:

void asignar_dato(Dato dat): Modifica el dato almacenado en el nodo.

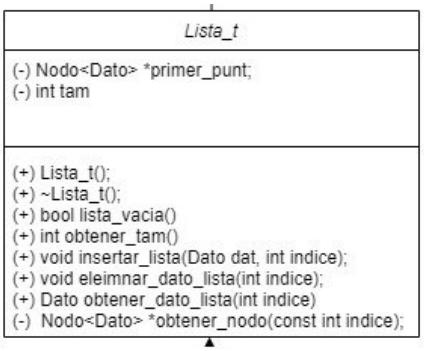
void asignar_siguiente_punt(Nodo *sig_punt): Modifica el puntero al siguiente nodo.

Geters:

Dato obtener_dato(): Devuelve el dato almacenado en el nodo.

Nodo *obtener_siguiente_nodo(): Obtiene el puntero al siguiente nodo.

Descripcion: La clase Lista_t es una clase template que representa una lista enlazada genérica. La lista puede almacenar cualquier tipo de dato (Dato) y ofrece una serie de funciones para insertar, eliminar y obtener datos de la lista.



Atributos:

primer_punt: Un puntero al primer nodo de la lista.

tam: El tamaño de la lista, es decir, la cantidad de nodos que contiene. **LISTA_T**

Métodos:

Constructores y destructores:

Lista_t(): Constructor que crea una lista vacía.

~Lista_t(): Destructor que libera la memoria asignada a todos los nodos de la lista.

Lista_se(const Lista_se &lista): Constructor de copia que crea una nueva lista con los mismos elementos que la lista original.

Seters:

void insertar_lista(Dato dat, int indice): Inserta un nuevo nodo con el dato especificado en la posición indicada por el índice.

void eleimnar_dato_lista(int indice): Elimina el nodo en la posición indicada por el índice.

Geters:

bool lista_vacia(): Devuelve true si la lista está vacía, false en caso contrario.

int obtener_tam(): Devuelve el tamaño de la lista.

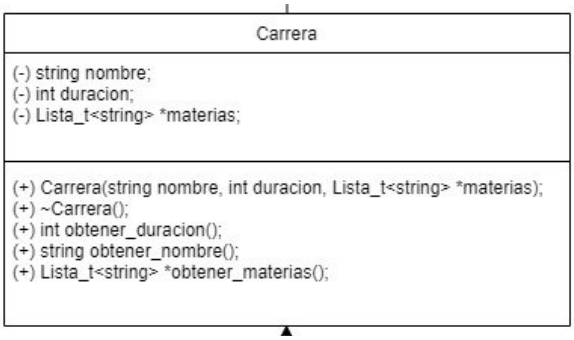
Funciones de consulta:

Dato obtener_dato_lista(int indice): Devuelve el dato almacenado en el nodo en la posición indicada por el índice.

Funciones auxiliares:

Nodo<Dato> *obtener_nodo(const int indice): Devuelve un puntero al nodo en la posición indicada por el índice.

Descripcion: La clase Carrera representa una carrera universitaria. La clase tiene información sobre el nombre de la carrera, la duración en años y una lista de las materias que se cursan en la carrera.



Atributos:

nombre: El nombre de la carrera.

duracion: La duración de la carrera en años.

materias: Un puntero a una lista de strings que contiene las materias de la carrera.

Métodos:

Constructores y destructores:

Carrera(string nombre, int duracion, Lista_t<string> *materias): Constructor que crea una nueva instancia de la clase Carrera con los valores especificados.

~Carrera(): Destructor que libera la memoria asignada a la clase.

Seters:

void insertar_lista(Dato dat, int indice): Inserta un nuevo elemento con el dato especificado en la posición indicada por el índice.

void eleimnar_dato_lista(int indice): Elimina el elemento en la posición indicada por el índice.

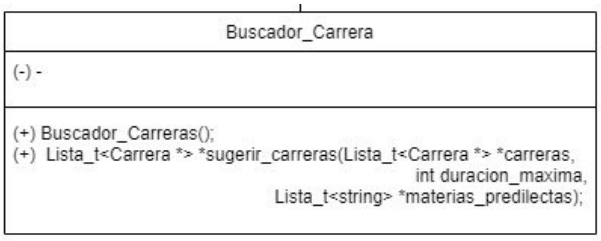
Geters:

int obtener_duracion(): Devuelve la duración de la carrera.

string obtener_nombre(): Devuelve el nombre de la carrera.

Lista_t<string> *obtener_materias(): Devuelve un puntero a la lista de materias de la carrera.

Descripcion: La clase Buscador_Carreras se encarga de buscar carreras que coincidan con las preferencias del usuario. La clase toma como entrada una lista de carreras, una duración máxima y una lista de materias predilectas, y devuelve una lista de las carreras que cumplen con las siguientes condiciones:



BUSCADOR_CARRERA

- Tienen una duración menor o igual a la duración máxima.
- Contienen al menos 3 de las materias predilectas.

Atributos:

- **No posee**

Métodos:

Constructores y destructores:

Buscador_Carreras(): Constructor por defecto que crea una nueva instancia de la clase.

Lista_se(const Lista_se &lista): Constructor de copia que crea una nueva lista con los mismos elementos que la lista original.

Funciones de busqueda:

Lista_t<Carrera *> *sugerir_carreras(Lista_t<Carrera *> *carreras, int duracion_maxima, Lista_t<string> *materias_predilectas): Esta función es la principal de la clase. Busca en la lista de carreras las que cumplen con las condiciones especificadas y devuelve una lista con las carreras que coinciden.