*TDA: ARBOL BINARIO DE BÚSQUEDA*

***ABB:***

Archivo principal del programa. Contiene los métodos más genéricos del árbol que permiten realizar las operaciones elementales y principales del programa. El árbol está compuesto por un solo atributo de tipo nodo\_abb que es la raíz del árbol, que representa el primer elemento. Luego cuando describamos la clase nodo\_abb se comprenderá mejor el funcionamiento del mismo

Atributos:

- Nodo\_abb\* raiz;

Métodos:

//Descripcion: Constructor sin parametros

//Pre:---

//Post: Se creo un arbol vacio

Abb();

//Descripcion: Constructor con parametros

//Pre: El parametro pasado debe ser valido

//Post: Se creo un arbol con un elemento

Abb(Nodo\_abb\* nueva\_raiz);

//Pre:El objteto debe existir

//Post:Se devuelve la raiz del arbol

Nodo\_abb\* obtener\_raiz();

//Pre: El parametro pasado debe ser valido.El objteto debe existir

//Post: Se modifico la raiz del arbol por el valor pasado por parametro

void asignar\_raiz(Nodo\_abb\* nueva\_raiz);

//Pre: El arbol debe existir

//Post: Indica si el arbol esta vacio o no

bool es\_vacio();

//Pre: EL arbol debe existir y el parametro debe ser un puntero valido

//Post: Se agrego el nodo al arbol

void agregar(Nodo\_abb\* nodo\_a\_agregar);

//Pre: EL arbol debe existir y el parametro debe ser un puntero valido

//Post: Se eliminó el nodo al arbol

void borrar(Nodo\_abb\*);

//Descripcion: Destructor

//Pre: El objeto debe existir

//Post: El objeto es destruido

~Abb();

***Nodo\_abb:***

El nodo es quien almacena el objeto que se crea a partir de los datos leídos del archivo de prueba o bien los que carga el usuario por consola. En este caso, la clase contiene 2 atributos que representan a los hijos izquierdo y derecho de la raíz del árbol, y a su vez de ellos mismo y así sucesivamente hasta llegar a un punto donde ambos son nulos, lo que representa el fin del árbol

Atributos:

- Nodo\_abb\* hijo\_izq

- Nodo\_abb\* hijo\_der

- string\* telefono

- Cliente\* cliente

Métodos:

//Descripcion: Constructor sin parametros

//Pre:---

//Post: Crea un objeto Nodo

Nodo\_abb();

//Descripcion: Constructor con parametros

//Pre:---

//Post: Crea un objeto Nodo

Nodo\_abb(Nodo\_abb\* nuevo\_hijo\_izq,Nodo\_abb\* nuevo\_hijo\_der,string nuevo\_telefono,Cliente\* nuevo\_cliente);

//Pre: El objeto debe existir

//Post: Devuelve el nodo del hijo derecho

Nodo\_abb\* obtener\_hijo\_der();

//Pre: El objeto debe existir

//Post: Devuelve el valornodo del hijo izquierdo

Nodo\_abb\* obtener\_hijo\_izq();

//Pre: El objeto debe existir

//Post: Devuelve el numero de telefono

string obtener\_telefono();

//Pre: El objeto debe existir

//Post: Devuelve el objeto cliente

Cliente\* obtener\_cliente();

//Pre: El objeto debe existir

//Post: Se modificó el nodo del hijo derecho

void asignar\_hijo\_der(Nodo\_abb\* nuevo\_hijo\_der);

//Pre: El objeto debe existir

//Post: Se modificó el nodo del hijo izquierdo

void asignar\_hijo\_izq(Nodo\_abb\* nuevo\_hijo\_izq);

//Pre: El objeto debe existir

//Post: Se modificó el número de teléfono

void asignar\_telefono(string nuevo\_telefono);

//Pre: El objeto debe existir

//Post: Se modifico el objeto cliente

void asignar\_cliente(Cliente\* nuevo\_cliente);

//Destructor

~Nodo\_abb();

***Lista***:

Contiene los métodos más genéricos de la lista, que permiten realizar las operaciones elementales y principales del programa. La lista incluye un puntero de tipo nodo, que apunta al primer elemento de la misma y el Tamanio de esta. A medida que se realicen operaciones, estos atributos se irán actualizando, y los puntos apuntarán al nuevo elemento que corresponda. A diferencia de la lista programada anteriormente, en este caso utilizamos plantillas que nos permiten almacenar distintos tipos de datos en la lista sin necesidad de utilizar un puntero genérico de tipo void. Para ello se denomina como un tipo abstracto “Dato” que varía según la necesidad en la función en la cual sea implementado

Atributos:

- Nodo\* primero;

- int tamanio;

Metodos:

//Definición de la plantilla: template <typename Dato>

//Pre: Debe existir un nodo en la posicion pasada por parametro

//Post: Cambia el nodo pasado por parametro por el nodo de la posicion pasada por parametro

- void cambiar\_nodo(Nodo\* nodo,int pos);

//Pre: Debe existir el nodo en esa posicion

//Post: Se va a retirar el nodo de esa posicion

- Nodo<Dato>\* retirar(int pos);

//Descripcion: Constructor

//Pre:---

//Post: Se crea el objeto Lista

+ Lista();

//Pre: El objeto debe existir

//Post: Devuelve el valor de tamanio

+ int obtener\_tamanio();

//Pre: El objeto debe existir, pos debe ser mayor a cero y menor o igual a tamanio

//Post: Se agrego el objeto d a la lista en la posicion pos

+ void agregar(Dato\* d, int pos);

//Pre: El objeto debe existir, pos debe ser mayor a cero y menor o igual a tamanio

//Post: Se eleimino de la lista el objeto en la posicion pos

+ void eliminar(int pos);

//Pre: El objeto debe existir

//Post: Devuelve true si la lista esta vacia y false en caso contrario

+ bool es\_vacia();

//Pre: El objeto debe existir,pos debe ser mayor a cero y menor o igual a tamanio

//Post: Devuelve el elemento de la lista en esa posicion.

+ Dato\* consultar(int pos);

//Pre: El objeto debe existir,pos debe ser mayor a cero y menor o igual a tamanio

//Post: devuelve el nodo enn la posicion pos de la lista

+ Nodo<Dato>\* obtener\_nodo(int pos);

//Pre: El objeto debe existir,pos1 y pos 2 deben ser mayores a cero y menores o iguales a tamanio

//Post: intercambio los nodos en pos1 y pos 2

+ void swap(int pos1, int pos2);

//Pre: El objeto debe existir

//Post: La lista se encuentra ordenada por numero de legajo de los trabajadores

+ void ordenar();

//Pre:El objeto debe existir

//Post:Devuelve verdadero si el dato esta en la lista o falso en caso contrario

bool esta(Dato\* dato\_a\_buscar);

//Descripcion: Destructor

//Pre:El objeto debe existir

//Post:EL objeto es destruido

~Lista();

***Nodo***:

El nodo es quien almacena el objeto que se crea a partir de los datos leídos del archivo de prueba o bien los que carga el usuario por consola, para ello contiene un puntero de tipo “Dato” (que representa a la plantilla) y otro puntero “Siguiente” que apunta al próximo elemento de la lista, dando origen así a lo que llamamos lista simplemente enlazada.

Atributos:

- Dato\* elemento;

- Nodo\* psig;

Metodos:

//Descripcion: Constructor conu un parametro

//Pre:---

//Post: El objeto es creado con los el valor d para elemento y le asigna a psig el valor por defecto

+ Nodo(Dato\* d);

//Descripcion: Constructor con parametros

//Pre:---

//Post: Crea un objeto Nodo

+ Nodo(Dato\* d, Nodo\* psig);

//Pre: El objeto debe existir

//Post: Se modificó el valor de elemento

+ void asignar\_elemento(Dato\* d);

//Pre: El objeto debe existir

//Post: Se modifico el valor de psig

+ void asignar\_siguiente(Nodo\* psig);

//Pre: El objeto debe existir

//Post: Devuelve el valor de elemento

+ Dato\* obtener\_elemento();

//Pre: El objeto debe existir

//Post: Devuelve el valor de siguiente

+ Nodo\* obtener\_siguiente();

//Descripcion: Destructor

//Pre: El objeto debe existir

//Post: El objeto es destruido

+ ~Nodo();

***Cliente:***

Superclase del programa, de la cual las clases individuo y familia heredan sus métodos y atributos. Representa al cliente que se almacena en el nodo del Abb.

Atributos:

- bool alta;

- string\* legajo;

- int precio\_base;

- char tipo;

Metodos:

//Pre:---

//Post: Crea un objeto cliente

Cliente();

//Descripcion: Constructor con parametros

//Pre:---

//Post: Crea un objeto cliente

Cliente(bool nueva\_alta, string nuevo\_legajo, int nuevo\_precio\_base, char nuevo\_tipo);

//Pre: El objeto debe existir

//Post: Devuelve si esta de alta o no

bool obtener\_alta();

//Pre: El objeto debe existir

//Post: Devuelve el numero de legajo

string obtener\_legajo();

//Pre: El objeto debe existir

//Post: Devuelve el precio base

int obtener\_precio();

//Pre: El objeto debe existir

//Post: Devuelve el tipo cliente siendo F familia e I individuo

char obtener\_tipo();

//Pre: El objeto debe existir

//Post: Se modifica el estado de alta

void asignar\_alta(bool nueva\_alta);

//Pre: El objeto debe existir

//Post: Se modifica el numero de legajo

void asignar\_legajo(string nuevo\_legajo);

//Pre: El objeto debe existir

//Post: Devuelve el precio base del cliente

int obtener\_precio\_base();

//Pre: El objeto debe existir

//Post: Devuelve el tamanio del cliente que representa la cantidad de miembros

virtual int obtener\_tamanio\_cliente();

//Pre: El objeto debe existir

//Post: Devuelve los nombres/nombre del cliente

virtual void mostrar\_cliente();

//Descripcion:Destructor

//Pre:El objeto debe existir

//Post:El objeto es destruido

virtual ~Cliente();

***Individuo:***

Clase heredera de cliente, que representa a una persona específica, que también es un cliente

Atributo:

- string\* nombre

Metodos:

//Descripcion: Constructor con parametros

//Pre:---

//Post: Crea un objeto individuo

Individuo(string\* nuevo\_nombre,bool nueva\_alta,string nuevo\_legajo, int nuevo\_precio\_base, char nuevo\_tipo);

//Pre: El objeto debe existir

//Post: devuelve el nombre del individuo

string obtener\_individuo();

//Pre: El objeto debe existir

//Post: Modifica el nombre del individuo

void asignar\_individuo(string\* nuevo\_nombre);

//Pre: El objeto debe existir

//Post: Se devuelve el tamanio del cliente

int obtener\_tamanio\_cliente();

//Pre: El objeto debe existir

//Post: Muestra los nombres del cliente/clientes

void mostrar\_cliente();

//Descripcion: Destructor

//Pre: El objeto debe existir

//Post: El objeto es destruido

~Individuo();

***Familia:***

Clase heredera que representa a un conjunto de individuos (mas de 1) que también se clasifica como un cliente

Atributos:

- Lista<string>\* familiares

Metodos:

//Descripcion: Constructor con parametros

//Pre:---

//Post: Crea un objeto familia

Familia(Lista<string>\* nueva\_familiares,bool nueva\_alta,string nuevo\_legajo, int nuevo\_precio\_base, char nuevo\_tipo);

//Pre: El objeto debe existir

//Post: Imprime por pantalla la familia

void mostrar\_familia();

//Pre: El objeto debe existir

//Post: Modifica la lista de famlia

void asignar\_familia(Lista<string>\* nueva\_familiares);

//Pre: El objeto debe existir

//Post: Se devuelve el tamanio del cliente

int obtener\_tamanio\_cliente();

//Pre: El objeto debe existir

//Post: Muestra los nombres del cliente/clientes

void mostrar\_cliente();//Nuevo mostrar\_familia.

//Descripcion: Destructor

//Pre: El objeto debe existir

//Post: El objeto es destruido

~Familia();