*TDA: LISTA POLIMÓRFICA*

Lista:

Archivo principal del programa. 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.

Atributos:

- Nodo\* primero;

- int tamanio;

Metodos:

//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\* 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(Trabajador\* 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.

+ Trabajador\* 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\* 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();

//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 a la clase “Trabajador” y otro puntero “Siguiente” que apunta al próximo elemento de la lista, dando origen así a lo que llamamos lista simplemente enlazada.

Atributos:

- Trabajador\* 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 (Trabajador\* d);

//Descripcion: Constructor con parametros

//Pre:---

//Post: Crea un objeto Nodo

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

//Pre: El objeto debe existir

//Post: Se modificó el valor de elemento

+ void asignar\_elemento(Trabajador\* 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

+ Trabajador\* 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();

Trabajador:

Clase que representa el corazón del programa, aquí se crea el espacio de memoria dinámica en donde se almacenan en distintas variables todos los datos correspondientes a cada trabajador. Esta se considera una superclase de “Jornalero”, “Consultor” y “Empleado” ya que estas heredan de forma pública los métodos y atributos de “Trabajador”.

Atributos:

- char \*tipo;

- int \*legajo;

- string \*nombre;

- int \*paga;

- bool \*alta;

- int \*sueldo\_liquidado;

Métodos:

//Descripcion:Constructor

//Pre:---

//Post: Se crea el objeto Trabajador

+ Trabajador();

//Descripcion: Constructor con parametros

//Pre: Tanto nueva\_paga como nuevo\_legajo deben ser enteros mayores a cero

//Post: Crea un objeto trabajador,asignando a sus atributos los parametros pasados

+ Trabajador (char nuevo\_tipo,int nuevo\_legajo, string nuevo\_nombre, int nueva\_paga, bool nueva\_alta);

//Pre: El objeto debe existir

//Post: Devuelve el valor del atributo tipo

+ char obtener\_tipo();

//Pre: El objeto debe existir

//Post: Devuelve el valor del atributo alta

+ bool obtener\_alta();

//Pre: El objeto debe existir

//Post: Devuelve el legajo del trabajador

+ int obtener\_legajo();

//Pre: El objeto debe existir

//Post: Devuelve el valor del sueldo liquidado

+ int obtener\_sueldo\_liquidado();

//Pre: El objeto debe existir

//Post: Devuelve el nombre del trabajador

+ string obtener\_nombre();

//Pre: El objeto debe existir

//Post: Devuelve la paga del trabajdor,es decir el valor leido del archivo

+ int obtener\_paga();

//Pre: El objeto debe existir

//Post: Se modifica el valor del atributo alta por nueva\_alta

+ void asignar\_alta(bool nueva\_alta); //No agregamos los otros asignar porque suponemos que solo se ingresan datos por la lectura del archivo

//Pre: El objeto debe existir

//Post: Se modifica el atributo sueldo\_liquidado

+ void asignar\_sueldo\_liquidado(int nuevo\_sueldo\_liquidado);

//Pre: El objeto debe existir

//Post: Devuelve el valor del sueldo liquidado del trabajador

+ virtual void liquidar\_sueldo(){} // Edit para que ande, sino me daba error en vtable

//Pre: El objeto trabajador debe existir,

//Post: Imprime los atributos nombre,legajo,tipo y sueldo a cobrar del trabajador

+ void a\_cadena();

//Descripcion: Destructor

//Pre: El objeto debe existir

//Post: El objeto es destruido

+ virtual ~Trabajador();

Jornalero:

Subclase de trabajador, que hereda de ella sus métodos y atributos

Atributos:

- int \*dias\_trabajados;

Metodos:

//Descripcion: Constructor

//Pre:---

//Post: Se crea el objeto Jornalero

+ Jornalero();

//Descripcion: Constructor con parametros

//Pre: Los parametros de tipo entero deben ser mayores o iguales a cero

//Post: Se crea el objeto Jornalero

+ Jornalero (char nuevo\_tipo,int nuevo\_legajo, string nuevo\_nombre, int nueva\_paga, bool nueva\_alta, int nueva\_dias\_trabajados);

//Pre: El objeto debe estar creado

//Post: Devuelve la cantidad de dias trabajados

+ int obtener\_dias\_trabajados();

//Pre: El objeto debe estar creado

//Post: Devuelve el sueldo liquidado

+ void liquidar\_sueldo();

//Descripcion: Desctructor

//Pre: El objeto debe estar creado

//Post: Destruye el objeto Jornalero

+ ~Jornalero();

Empleado:

Subclase de trabajador, que hereda de ella sus métodos y atributos

Atributos:

- int \*llegadas\_tarde;

- int \*ausencias;

Metódos:

//Descripcion: Constructor

//Pre:---

//Post: Se crea el objeto Empleado

+ Empleado ();

//Descripcion: Constructor con parametros

//Pre: Los parametros de tipo entero deben ser mayores o iguales a cero

//Post: Se crea un objeto Empleado

+ Empleado(char nuevo\_tipo,int nuevo\_legajo, string nuevo\_nombre, int nueva\_paga, bool nueva\_alta, int nueva\_llegadas\_tarde, int nueva\_ausencias);

//Pre: El objeto debe estar creado

//Post: Devuelve la cantidad de llegadas tarde que tiene el Empleado

+ int obtener\_llegadas\_tarde();

//Pre: El objeto debe estar creado

//Post: devuelve la cantidad de ausencias quetiene el Empleado

+ int obtener\_ausencias();

//Pre: El objeto debe estar creado

//Post: Devuelve el sueldo liquidado del Empleado

+ void liquidar\_sueldo();

//Descripcion: Destrcutor

//Pre: El objeto debe estar creado

//Post: El objeto es destruido

+ ~Empleado();

Consultor:

Subclase de trabajador, que hereda de ella sus métodos y atributos

Atributos:

- int \*horas\_catedra;

- int \*horas\_descontadas;

Metodos:

//Descripcion: Constructor

//Pre:---

//Post: Se crea el objeto Consultor

Consultor();

//Descripcion: Constructor con parametros

//Pre: Los parametros de tipo entero deben ser mayores o iguales a cero

//Post: Se crea el objeto Consultor

+ Consultor(char nuevo\_tipo,int nuevo\_legajo, string nuevo\_nombre, int nueva\_paga, bool nueva\_alta, int nueva\_horas\_catedra, int nueva\_horas\_descontadas);

//Pre: El objeto debe estar creado

//Post: Devuelve la cantidad de horas catedra trabajadas

+ int obtener\_horas\_catedra();

//Pre: El objeto debe estar creado

//Post: Devuelve la cantidad de horas a descontar

+ int obtener\_horas\_descontadas();

//Pre: El objeto debe estar creado

//Post: Devuelve el sueldo liquidado

+ void liquidar\_sueldo();

//Descripcion: Destructor

//Pre: El objeto debe estar creado

//Post: El objeto es destruido

+ ~Consultor();

Funciones:

Archivo de funciones auxiliares programado de manera estructurada para el manejo y puesta en común de todas las clases anteriores. Permite realizar algunas operaciones y acortar unas cuantas líneas el main, con el fin de lograr una mejor modularización en el programa

//Pre:---

//Post:Imprime por pantalla el menu de comandos

void menu();

//Pre: El parametro pasado debe ser valido

//Post: Devuelve el tipo de trebajador leido

char leer\_tipo\_trabajador(ifstream &archivo);

//Pre: La lista debe existir y el archivo se debe poder abrir

//Post: La lista va a estar cargada por los datos leidos en el archivo

void cargar\_lista(ifstream &archivo,Lista\*);

//Pre:La lista debe estar creada

//Post:La lista esta ordenada por numero de legajo

void ordenar\_lista(Lista\* lista);

//Pre:La lista debe existir y el legajo a buscar debe ser un entero positivo

//Post:Devuelve la posicion del legajo buscado en la Lista o -1 si el legajo buscado no se encuentra en esta

int buscar\_legajo(int legajo\_a\_buscar, Lista\* lista);

//Pre:La lista debe existir y pos\_legajo debe ser un entero mayor a cero y menor o igual al tamnio de la lista

//Post:El atributo alta del trabajdor es modificado

void cambiar\_alta(int pos\_legajo,Lista\* lista ,bool cond);

//Pre:Modo vale A si busca maximo o B si busca minimo.Ademas la lista debe existir

//Post:Devuelve el sueldo maximo o minimo de la lista

int buscar\_sueldo(char modo, Lista\* lista);