Reporte 1: TDA Lista simplemente ligada

Como fue indicado en classroom, el programa es una lista simplemente ligada que puede manejar datos de tipo string, y además por decisión personal decidí hacerlo en 3 archivos distintos, el cpp principal, y dos hpp, uno para la clase nodo, y otro para la clase y todos los métodos de la lista. Además, estas clases están hechas con los atributos sin encapsular.

nodo.hpp

El código lo explicaré empezando por el hpp del nodo, el cual también es el más corto y el que necesita menos explicación, pues se explica casi solo, lo único que veo importante mencionar en esta parte son los dos elementos que contiene un nodo, el cual es el elemento dato y el elemento siguiente, y que se declaran en los atributos de la clase, y se inicializan fuera de ella.

También, para guardar datos de tipo string en la lista, fue necesario indicar que el elemento “dato” era de tipo string, e inicializarlo con una cadena.

```
#ifndef NODO_H
#define NODO_H
#pragma once
using namespace std;

class nodo
{
public:
    string dato;
    nodo *siguiente;
    nodo();
    nodo(string e)
    {
        dato = e;
        siguiente = nullptr;
    };

private:
};
nodo::nodo()
{
```

```
    dato = '0'; // obj inicializar el objeto
    siguiente = nullptr;
}
#endif
```

listaSimple.hpp

Al inicio del archivo tenemos la declaración de la clase lista, con los punteros que apuntarán al inicio y al final de ella, además de todas las operaciones (métodos de la clase) que se pueden hacer en nuestra lista simplemente ligada, y ya que nuestra lista recibe datos de tipo string, estos métodos tienen declarado que van a recibir strings.

```
11  int tamano = 0;
12  class listaSimple
13  {
14  public:
15      listaSimple();
16      nodo *h; // Puntero que apunta a inicio
17      nodo *t; // Puntero que apunta a final
18      void inicializa();
19      void insertaInicio(string);
20      void insertarFinal(string);
21      void mostrarLista();
22      void tamanoLista();
23      void buscarElemento(string);
24      void eliminarElemento(string);
25      void eliminarLista(string &);
26
27  private:
28  };
```

Veo innecesario explicar a detalle cada operación de la lista, por lo que solo explicaré unas cuantas las cuales son más importantes, yendo por orden según está escrito en los métodos de la clase lista.

Inicializa

Este método es sencillo, simplemente inicializa h a nulo, no hace falta explicar más

Insertar inicio

Creamos un nodo y le asignamos el dato del usuario, después con un auxiliar apuntamos al que antes era el inicio de la lista, y ya que lo insertamos al principio,

decimos que h (el inicio) es el nodo que acabamos de insertar, por ultimo apuntamos el siguiente del nodo que acabamos del crear al 2do nodo de la lista, el cual es, aux1.

```
54      nodo *aux1 = h;
55      h = nuevo_nodo;
56      nuevo_nodo->siguiente = aux1;
```

Insertar final

```
73      while (aux1 != nullptr)
74      {
75          aux2 = aux1;
76          aux1 = aux1->siguiente;
77      }
78      aux2->siguiente = nuevo_nodo;
79      nuevo_nodo->siguiente = aux1;
```

Después de crear el nodo y asignarle el dato, aquí simplemente tenemos un ciclo while para recorrer la lista hasta el final, cuando llegamos al final y nos salimos de la lista, el aux2 va a estar justo en el ultimo nodo, por lo que procedemos a insertar el nuevo nodo después de aux 2, y apuntamos este al nuevo nodo.

Mostrar lista

Este es otro algoritmo sencillo así que no veo la necesidad de poner el código, solo es un algoritmo para recorrer toda la lista de inicio a fin como el que vimos en la operación pasado, solo que va imprimiendo el dato de cada nodo que pasa.

Tamaño lista

De nuevo recorremos toda la lista, pero ahora cada que avanzamos un nodo vamos agregando 1 a nuestro contador, así que al final nos dice por cuantos nodos pasó, es decir, el tamaño.

Buscar elemento

De nuevo, recorremos toda la lista, y si encontramos un nodo que sea igual al que el usuario pidió, le diremos que sí existe.

Eliminar elemento

Este es uno de los algoritmos más largos porque son 3 casos, primero tenemos que comprobar que la lista no esté vacía, y luego buscar el elemento con el algoritmo que ya conocemos para recorrer la lista, en caso de que no encontremos el elemento simplemente le decimos al usuario que ese elemento no existe, si lo encontramos tenemos que ver en donde está

Si se encuentra al incio

```
else if (anterior == NULL) // El primer elemento es el que se elimina
{
    h = h->siguiente;                                    // el inicio de la lista se cambi
    cout << "El elemento '" << n << "' ha sido borrado\n"; // Imprimir el elemento a borrar
    delete aux_borrar;
```

Apuntamos el inicio de la lista al 2do elemento, y borramos el elemento en el que estamos.

Si se encuentra en otra posición

```
else // El elemento que se elimina no es el primer elemento
{
    anterior->siguiente = aux_borrar->siguiente;
    cout << "El elemento '" << n << "' ha sido borrado\n";
    delete aux_borrar;
```

Tenemos un auxiliar llamado anterior que va un paso detrás del nodo que vamos a borrar, apuntamos este anterior a dentro de 2 nodos “saltándose” el que se va a borrar, para poder borrar el nodo sin perder los elementos de la lista.

Eliminar lista

Esta usa el mismo algoritmo para eliminar el 1er elemento de la lista, solo que lo hace en bucle hasta que la lista se quede sin elementos, es decir, vacía

Variables

Durante todo el programa se usan principalmente 3 variables, 2 auxiliares, las cuales creo que no es necesario especificar qué hacen, pues solo son variables temporales para ayudar a la ejecución del algoritmo, y la variable *n,* la cual es una variable local de cada función, sin embargo en todas funciona de la misma manera, y es la variable que contiene el elemento a agregar escrito por el usuario, es por eso que la podemos ver en funciones donde el usuario ingresa un elemento, como en el caso de la imagen anterior con la función que inserta al inicio, pero no en la función que muestra la lista, pues aquí el usuario no ingresa ningún dato.

```
82  void listaSimple::mostrarLista()
83  {
84      cout << tamano << " elementos en la lista: " << endl;
85      nodo *actual = new nodo(); // Creamos nodo actual para saber
86      actual = h;                // Apuntar el nodo actual al inici
87
88      while (actual != nullptr) // Mientras sigamos apuntando a un
89      {
90          cout << actual->dato << " - "; // Imprimir el dato en el
91          actual = actual->siguiente;    // Recorrer un nodo
92      }
93      cout << endl;
94  }
```

Menu para lista.cpp

Por ultimo, voy a explicar de manera rápida el cpp principal, ya que simplemente es el menú que va a ver el usuario

En la línea 6 está el prototipo para la función del menú que veremos en unos momentos, y en la línea 7 creamos una nueva lista en la cual guardaremos los datos del usuario.

```
6   void menu();
7   listaSimple *lista = new listaSimple();
8
9   int main()
10  {
11      menu();
12      system("PAUSE");
13      return 0;
14  }
15
```

```
16  void menu()
17  {
18      int opc = 1;
19      string dato;
20      while (opc != 0)
21      {
22          system("cls");
23          cout << "\tMenu\n";
24          cout << "1. Insertar al inicio\n";
25          cout << "2. Insertar al final\n";
26          cout << "3. Mostrar lista\n";
27          cout << "4. Buscar un elemento\n";
28          cout << "5. Eliminar elemento\n";
29          cout << "6. Vaciar lista\n";
30          cout << "7. Mostrar tamano lista\n";
31          cout << "0. Salir\n\n";
32          cin >> opc;
33          switch (opc)
34          {
35          case 1:
36              cout << "Ingrese un dato: ";
37              cin >> dato;
38              lista->insertaInicio(dato);
39              break;
40          case 2:
41              cout << "Ingrese un dato: ";
42              cin >> dato;
43              lista->insertarFinal(dato);
44              break;
45          case 3:
46              lista->mostrarLista();
47              break;
```

Pasando a la función del menú, simplemente está lo que se imprime en pantalla, el usuario ingresa el numero de la opción a la que quiere acceder, y con un switch lo redirigimos a donde se hace la llamada de ese método en la clase lista, por ejemplo el caso 1 que es para insertar al inicio, llama al método *insertarInicio.*

(Para esto es necesario incluir el hpp de lista en el archivo cpp, eso no lo muestro en foto ya que es algo un poco obvio).

Por ultimo, en el menú se le dice al usuario que la opción 0 es para salir del programa, por lo que el programa se ejecutará sin parar hasta que el usuario elija esta opción gracias al while de la línea 20, además de que no

se imprimirá de manera desordenada en la pantalla por el cls que está en la línea 22, y que borra de la pantalla todo lo que estaba antes de imprimir el menú.

Conclusión

Ya que mi conclusión es de manera personal sobre el conocimiento adquirido, la escribiré de una manera más informal que el resto del reporte.

Aunque entiendo bien cada algoritmo de la lista simplemente ligada (de nuevo, prueba de esto es este programa), entiendo su función, lógica, etc, sigo pensando que es más sencillo hacer un arreglo que una lista (obviamente), y que salvo en contados casos específicos, puede que siempre sea más conveniente usar alguna alternativa a tener que codificar una estructura de datos, aún así, entiendo que es necesario aprender a cómo hacer todas estas estructuras de datos precisamente para esos casos específicos en los que sí sea más conveniente usar esas a buscar alguna alternativa.

Ya que ya entiendo las listas simplemente ligadas, creo que las listas doblemente ligadas se me harán más sencillas, pues simplemente es agregarle el factor del elemento anterior al nodo, además que ya tengo un conocimiento base sobre estructuras de datos, a diferencia de cuando empezó la materia que no conocía nada.