**Marco teórico**

**Listas simplemente enlazadas**

Decimos que las listas simplemente enlazadas son estructuras de datos que son semejantes a los arreglos, solo que el acceso a los elementos dentro de las mismas no se hace mediante un indice, si no que se realizan mediante el uso de punteros. Siendo así su asignación en memoria hecha durante la ejecución.

En una lista los elementos son contiguos en lo que concierne al enlazado.

Al contrario, en los arreglos los elementos están contiguos en la memoria, mientras que en la listas se encuentran dispersos. Los enlaces entre los elementos de las listas se hacen mediante punteros. El puntero siguiente del ultimo elemento tiene que apuntar a NULL que decimos que es el final de la lista.

Para accesar a un elemento, la lista es recorrida comenzando desde el inicio, y gracias al puntero que apunta al siguiente nodo, se puede cambiar al siguiente elemento que se encuentra en esta misma. El desplazamiento se realiza en una sola dirección, del primer al ultimo elemento.

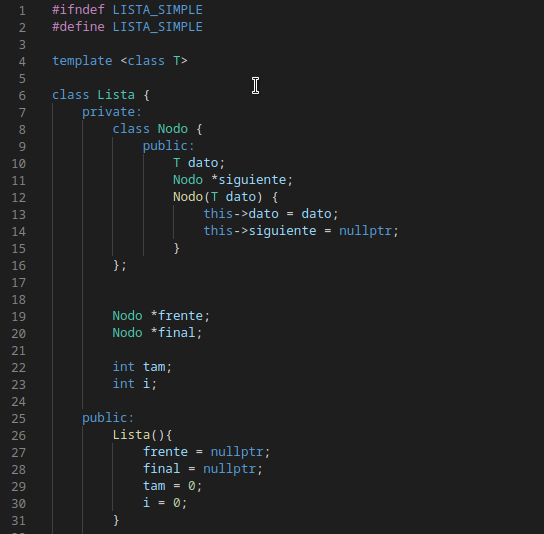
**(Esto va en las referencias) Fuente:** Guerrero, N. (2019, 31 diciembre). *La lista enlazada simple*. ProgramaEnLinea. https://www.programaenlinea.net/la-lista-enlazada-simple/

**Desarrollo**

**Listas simplemente enlazadas**

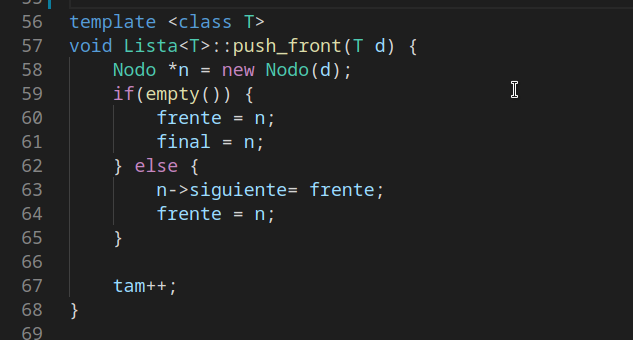
Primero se creó un archivo que usaremos como librería para nuestro main, en este archivo utilizado como librería se crea una clase nodo que contiene a dato que es de tipo T que significa que almacenara el tipo de dato que el usuario agregue, hacemos un apuntador de nodo que será siguiente y creamos un Nodo al frente y uno al final, se crea una variable de tipo entero que almacenará el tamaño de nuestra lista y una variable i que sera utilizada para nuestras funciones donde queramos hacer cambios en la posición deseada.

Nuestros punteros de frente y final los apuntamos a nulo y nuestra variable de tam la igualamos a cero que significa que nuestra lista esta vaciá.

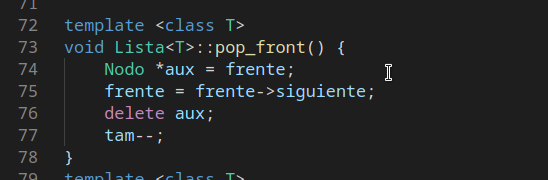


Para el push\_front() primero se crea un nuevo nodo con apuntador en n que sera un nuevo nodo con el dato que el usuario agrega, luego verifica si la lista esta vaciá para si este caso se cumple va asignar el frente a n y el final también apuntará a n.

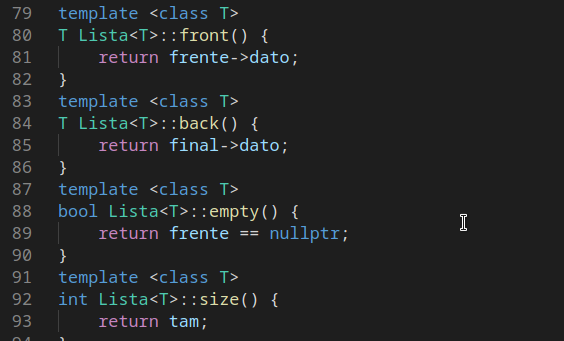
De lo contrario le dira que el siguiente apuntador de n sea igual al frente y el frente ahora sea el nuevo nodo n, y aumentara el tamaño en tam.



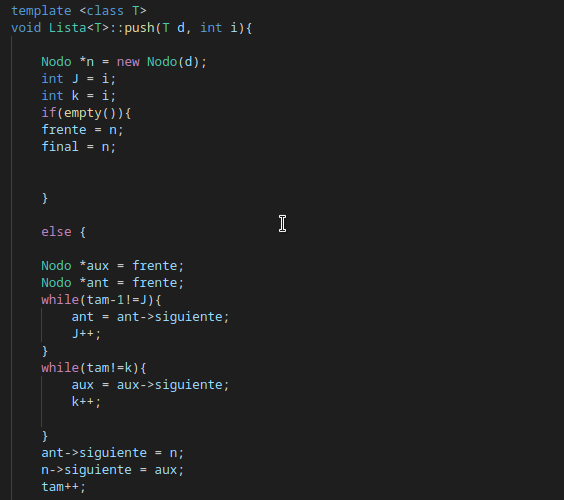
Para el pop\_front() se crea un nodo auxiliar el cual apuntará a frente y después el apuntador frente le decimos que sea igual a su siguiente. Procedemos a eliminar el nodo auxiliar para liberar memoria y decrementamos el tamaño de la lista en tam.



Para las siguiente funciones que son front(), back(), empty() y size() usando los apuntadores retornamos el dato que se encuentre como en el caso de front que nos muestre el dato que apunta frente, en back que nos apunte el dato de final, empty es una función de tipo booleano que nos regresara verdadero cuando el frente sea igual a nulo, y la función de size lo que hará es regresarnos el valor de tam.

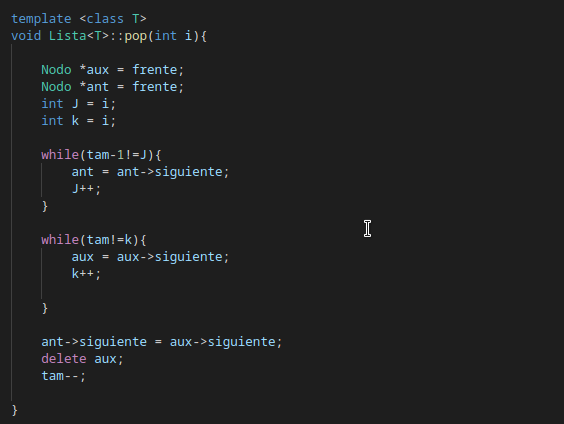


Para el push() que sera la función que agregará un elemento en la posición de i, se hace lo siguiente, creamos un nodo n que sera el nuevo nodo con el dato que se desea agregará, creamos dos variables J,k para el uso de nuestros whiles que serán asignados con el valor de i que se obtiene, hacemos la comprobación si la lista esta vaciá y hace lo mismo que en la anterior. En el caso contrario creamos un nodo auxiliar que apunte a frente y un nodo ant que será el que utilizaremos para recorrer la lista y obtener el nodo anterior al nodo que se encuentra en la posición, el primer while es para encontrar el nodo anterior que nuestro condicional es que mientras el tamaño de la lista -1 (para estar en el anterior a este) sea diferente de J que J vale i, el nodo de ant sea el mismo en su siguiente así hasta llegar al nodo anterior, con j++ para ir cambiando el valor del nodo y salga del bucle. Luego el segundo bucle es para que el nodo auxiliar apunte al nodo que remplazaremos en i. después el nodo anterior en su siguiente lo apuntaremos a n y que el nodo siguiente d n apunte al auxiliar y así habremos sobrepuesto el nodo en la posición i



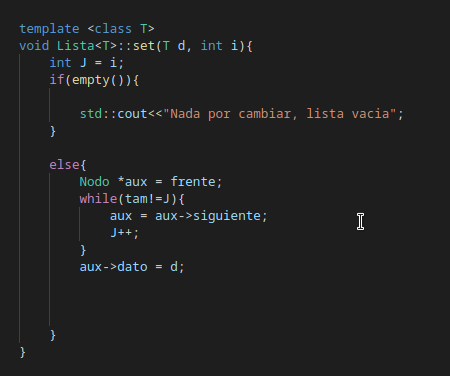
Para el Pop() que será la función que elimine el nodo en la posición deseada se crean de igual forma dos nodos que sea uno auxiliar y otro anterior que ambos apuntaran frente

Se utilizaran los mismos whiles que se utilizaron en la función anterior para tener el nodo anterior al de la posición y el nodo auxiliar apuntando al que sera eliminado despues de recorrer los whiles el nodo anterior a su siguiente sera igual al nodo auxiliar en su siguiente y procedemos a eliminar al nodo auxiliar y hacemos decremento en tamaño.

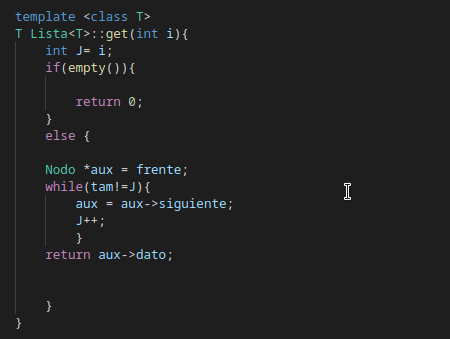


Para el método set() se crea una variable J que sera utilizado para el while y tendrá el mismo valor que i, se comprueba si la lista esta vaciá, si es así mostrara un mensaje que dirá que no hay nada por cambiar.

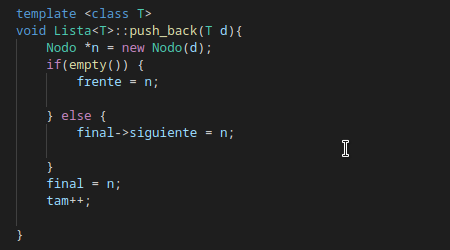
En el caso contrario se crea un nodo auxiliar que apunta a frente y se hace un while que recorre la lisa para llegar al dato que se encuentra en la posición i, luego le decimos que el nodo auxiliar en dato sea el dato que se cambiará.



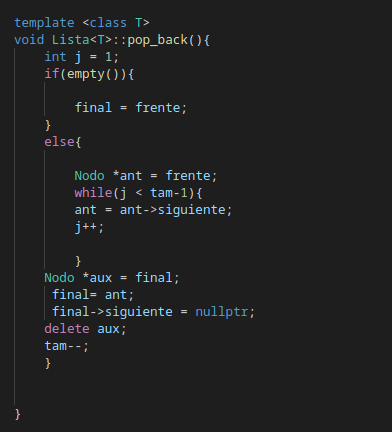
En la función get() la cual es la encargada de mostrarnos el dato que se encuentra en la posición i, se crea de igual forma la variable J que será asignada con el valor de i para el while primero comprueba que la lista no este vaciá, si es así retornara cero, en el caso contrario se crea un nodo auxiliar que apunta a frente y se hace un while que recorre a auxiliar para llegar a la posición deseada como en los anteriores bucles y después que termine nos retornará el dato al que apunta auxiliar.



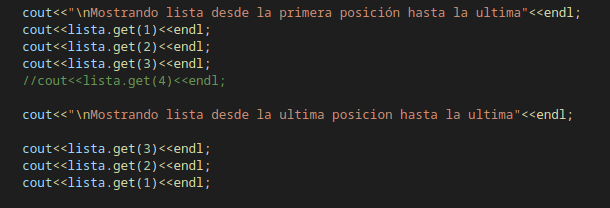
Para la función push\_back() se crea el nodo n que sera un nuevo nodo con el dato d, se comprueba que la lista no este vacia, de ser así el frente de la lista sera igual al nodo n en el caso contrario final en su siguiente apuntara al nodo n y final apuntara a n y aumentamos el tamaño en tam.



En la función pop\_back() se crea una variable j que sera igual a 1, si la lista esta vaciá final apuntara a frente en el caso contrario se crea un nodo anterior que apunte a frente y que con un while recorrerá la lista para llegar al nodo anterior al ultimo nodo, cumpliéndose la condicional de que mientras j sea menor al tam-1 o sea el anterior el nodo ant sea ant en su siguiente, saliendo del bucle, se crea un nodo auxiliar que apunte a final, después final lo apuntamos a ant y final en si siguiente lo apuntamos a nulo y eliminamos el nodo auxiliar y decrementamos tam.



Para recorrer la lista del principio a final, se usa la función get y la usamos en las posiciones que existan en nuestra lista y para hacerlo al contrario, escribimos el orden de igual forma



**Conclusiones**

Victor Hugo Estrada Ortega :

Esta practica me gustó mucho realizarla ya que nunca había hecho uso de punteros en la programación y siento que es muy útil y mucho más para actividades como estas, me imagino que no solo hay de aplicaciones esto y que hay miles de usos más que los punteros tienen, también disfrute mucho de programar porque en lo personal iba aprendiendo mientras lo iba haciendo y también aprendía a resolver los problemas que se me generaban y después de hacer que el programa corra es una gran satisfacción. Así como también se aplica el tema de Listas y haciendo este tipo de practicas ayudan a entender mejor los temas y su modo practico.