Reporte Individual Unidad I
Double-Ended Queues

Torres Colorado Juan Daniel*
*Ingeniería en Tecnologías de la Información
Universidad Politécnica de Victoria

***Resumen*—El presente documento tiene como próposito informar a detalle el proyecto individual de la Unidad I de la materia Lenguajes y Autómatas. El proyecto se centra en la implementación de estructuras de datos de colas dobles (deque) en el lenguaje de programación C++ y se utilizaron los principios de Programación Orientada a Objetos (POO) en la cual elaboró la implementación práctica de distintas variantes de deque; como el deque ADT, el deque STL, el deque con listas doblemente enlazadas y el deque con Stack ADT. El proyecto Double-Ended Queues implementa varias funciones y clases para colas dobles.**

I. Introducción

El proyecto de nombre "Double-Ended Queues [1]" fue desarrollado en el lenguaje de programación C++ [2] utilizando la Programación Orientada a Objetos (POO) [3]. Este proyecto fue designado por el profesor y tuvo lugar a finales de febrero del presente año. Además, con apoyo de parte de diversas fuentes proporcionadas por el profesor, tales como las clases impartidas en la materia, el libro "*Data Structure and Algorithms* [4]" e investigación externa, sentaron las bases para satisfacer las necesidades tanto de conocimiento como de practicidad para la elaboración de dicho proyecto.

II. Desarrollo Experimental

El desarrollo experimental que se llevo a lo largo de este proyecto fue con el motivo de desarrollar una aplicación de consola que sea capaz de demostrar el funcionamiento de la cola doble (Double-Ended Queues, tambien conocidas como "*deque*") [5]. Además, el sistema operativo que se usó para el desarrollo de esta aplicación de consola, fue del software Zorin OS [6], la cual es una distribución de Linux que ofrece una interfaz similar a Windows.

Para iniciar, se hizo una investigación exhaustiva (con los materiales anteriormente ya mencionados) sobre el cómo se utiliza las colas dobles, seguido de pruebas para el manejo de este tipo de dato estructurado (ADT) [7] y se llevaron acabo distintas practicas y consultas para llevar un mejor entendimiento y comprensión acerca de cómo es que trabaja dicho tipo de dato.

Tras haber llevado a cabo la comprensión de la funcionalidad (tanto de metodos como de atributos) del deque, se llevo a cabo otra investigación acerca de los "*templates*" [8]. Esto con el propósito de trabajar con deque's de la manera más eficiente y óptima para lograr demostrar cómo es que trabajan los deque's ADT almacenando tres tipos de datos primitivos (elegidos personalmente): enteros, cadena de caracteres y flotantes. En la figura 2 (a) se muestra el menú principal que (entre otras opciones que se explicarán más adelante) contiene la opción de elegir uno de estos tres tipos de datos primitivos y proceder con la desmostración acerca de los deque's.

Las ideas principales que se desea enseñar en este proyecto son: demostrar cómo funciona el deque y como se pueden aplicar similarmente en otros tipos de estructura de datos. Por consiguiente, se llevará un manejo de una interfaz de consola capaz de trabajar con:

1. Métodos de deque ADT (e.g., insertFront(*e*), insertBack(*e*), front()...).
2. Métodos de deque STL
3. Deque con Listas Doblemente Enlistadas
4. Deque con Stack ADT

Exceptuando al deque ADT, para el desarrollo de cada elemento de la lista anterior, se acompletó el código que proporciona el libro *Data Structures and Algorithms in C++*. Además de ello, se implementó la actualización necesaria para cada ejemplo proporcionado, debido a que estos mismos cuentan sin mantenimiento previo, quedan (en ciertos casos) de alguna pequeña o gran manera, obsoletas para su implementación.

Para la implementación de este proyecto, se utilizó una computadora con un procesador Intel Core i7-8550U a 1.8GHz junto con 16 GB de RAM, cumpliendo las necesidades para el desarrollo y ejecución de este proyecto.

III. Resultados

Primeramente, en la figura 1 podemos observar un diagrama en la que demuestra la relación entre cada clase necesaria para este proyecto. Se realizó con el fin de llegar a obtener una mejor comprensión acerca de cómo es que el proyecto trabaja, siendo que el diagrama UML demuestra qué métodos y atributos forma en cada clase. Cabe aclarar que la clase "Aplicacion" realmente no forma parte de una clase real, sino, que es la encargada de ejecutar los procesos necesarios para ejectuar la aplicacion, en todo caso, se refiere el main.cpp. De manera general, para los deque ADT y STL se guardaran en una clase (debido a que siguen el mismo comportamiento, solo se "traduce" el nombre del método equivalente a STL). Observando esta imagen, podemos apreciar que para llevar una relación con todas las clases proporcionadas, no es necesario incrustarlas en el main, sino, incrustarlas en los lugares en donde se necesite llamar, es decir, la clase *DequeStack* llama a la clase *LinkedDeque* y esta misma llama a *DLinkedList* que a su vez llama a la clase *DNodo*. La relación de cada uno se

podría describir como un ”vocero” o ”dependiente”; es decir, si se quiere realizar una acción dentro de las listas doblemente enlistadas, entonces esta deberá hacer un recorrido entre las clases para llegar al nodo *DNode* en donde se quiera efectuar una acción. Por ejemplo, si quisiera hacer revisar desde mi clase *DequeStack* cuantos elementos tengo, entonces tendría que llamar al método *size()*, la cuál esta llama al método *size()* de *LinkedDeque* que a su vez llama al método *size()* de *DLinkedList* y esta última, me realiza el trabajo de tener el control de los elementos.

En las figuras 2, 3 y 4 muestran (además del menú principal)los metodos que ADT y STL tengan disponibles. Se podría considerar que ADT y STL son hermanos, ya que si algo se modifica en una, en la otra se modificará también y esto sucede debido a que se hizo una sola clase respectiva para ambos llamada *GenericalDeque*, en la cual STL está encargado de declarar los metodos ADT, dado que ambos tienen el mismo propósito y lo unico que los diferencia son sus nombres, entonces solo tendriamos que crear una ”media” entre ADT y STL; es decir, cuando se llama algún método ADT, STL está encargado de realizar dicho seguimiento.

Para la figura 5 simplemente se hace uso similar ya sease al deque ADT o STL. Sin embargo, cabe recalcar que a diferencia de estos, u otros, esta es la clase encargada de llevar a cabo todos los procesos en los nodos, ya sea que se modifique, elimine u otra acción.

Para terminar, en la última figura 6 se hace uso de Stack ADT, esta clase actua de manera similar a lo que una cola (queue) normal se comportaría. Sin embargo, no es una cola normal, debido a que, como su nombre lo menciona, esta almacena listas enlazadas (que a su vez, doblemente enlazada por la siguiente clase).

IV. Conclusión

En conclusión, este proyecto permitió al alumno adquirir un profundo entendimiento de las colas dobles, específicamente comprendiendo en que puede llegar a haber múltiples formas de crear esta estructura de dato y su implementación en C++. Se logró implementar una aplicación funcional por consola que demuestre el uso práctico de diferentes enfoques: desde el deque básico hasta variantes más complejas. Además, destacando la importancia de la Programación Orientada a Objetos y la flexibilidad que proporcionan los Templates en la manipulación de datos, permitió y motivó al alumno a seguir desarrollándose proporcionando experiencias valiosas en la resolución de problemas y la toma de decisiones en la implementación de estructuras de datos. Por última instancia, este proyecto contribuyó significativamente al crecimiento personal como la paciencia y llevar tranquilidad ante la resolución de conflictos, así también como conocimiento teórico y práctico en el ámbito de las estructuras de datos y la programación en C++, proporcionando una base sólida para abordar desafíos más avanzados en futuros proyectos.

Referencias

[1] Jitender Putnia. *Deque in Data Structure*. https://www.scaler.com/topics/dequeue-in-data-structure/. Consultado el 01-03-2024.

[2] Romain Juillet. *Essentials of Programming in C++*. https://www.bocasay.com/essentials-programming-c/. 2023.

[3] Kyle Herrity. *What is Object Oriented Programming (OOP)?* https://www.indeed.com/career-advice/career-development / what - is - object - oriented - programming. 2023.

[4] Michael T. Goodrich & Roberto Tamassia & David Mount. *Data Structures & Algorithms in C++*. Second Edition. 2011.

[5] Runestone. *¿Qué es una cola doble?* https://runestone.academy/ns/books/published/pythoned/BasicDS/QueEsUnaColaDoble.html. Consultado el 01-03-2024.

[6] Zorin OS. *Make your computer better*. https://zorin.com/os/. Consultado el 01-03-2024.

[7] Ankit Kochar. *Abstract Data Type (ADT)*. https://www.prepbytes.com/blog/data-structure/abstract-data-type-adt-in-data-structure/. 2023.

[8] cplusplus. *Templates in C++*. https://cplusplus.com/doc/oldtutorial/templates/. Consultado el 01-03-2024.

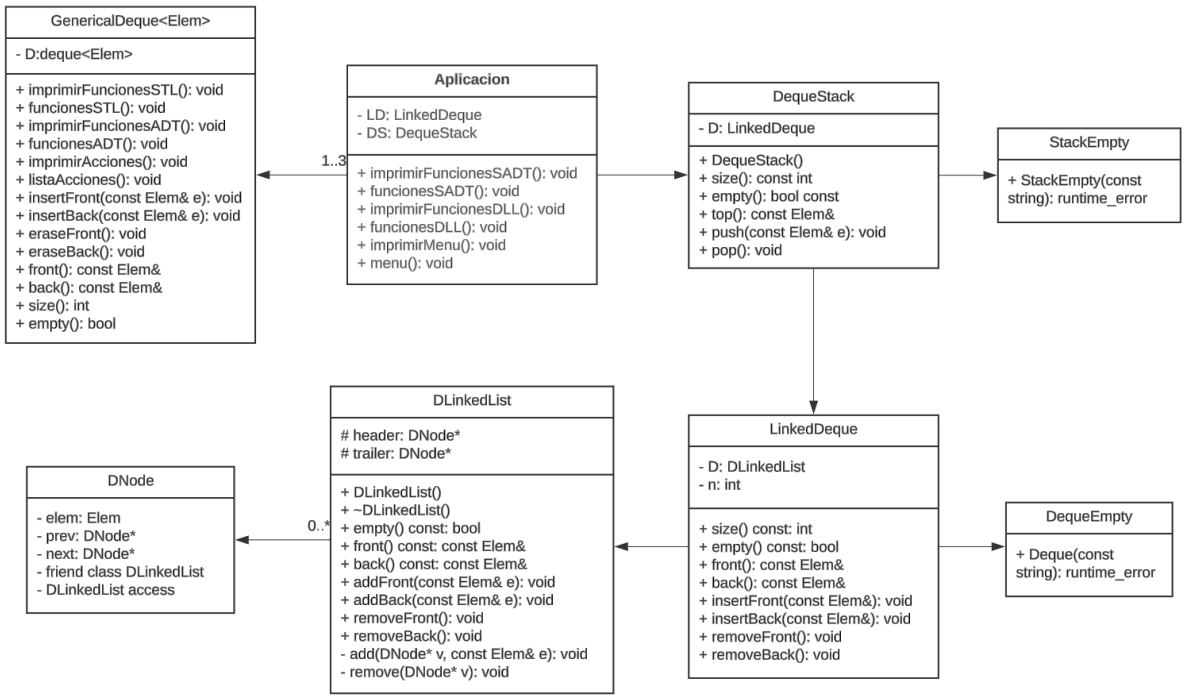
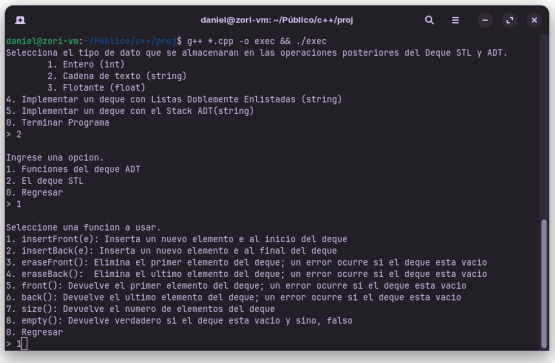
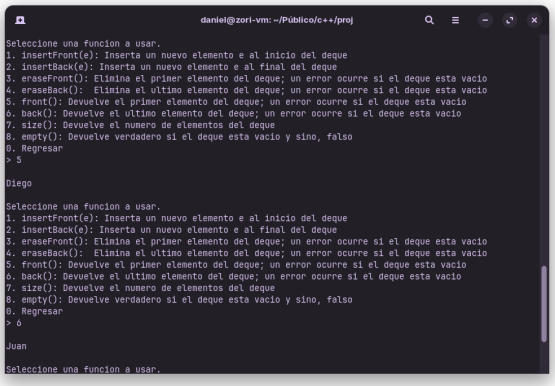


Figura 1: Relacion UML sobre la relacion entre clases



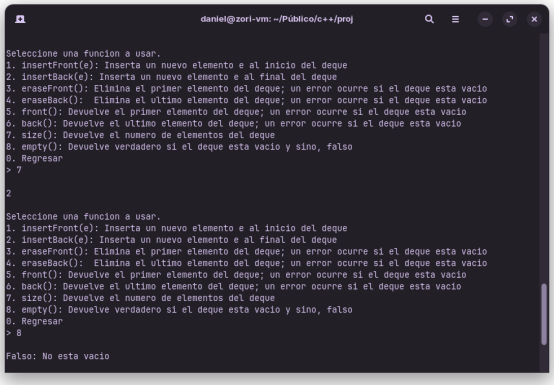
(a) Menu Principal

(b) Ingreso de elementos con *insertFront(e)*

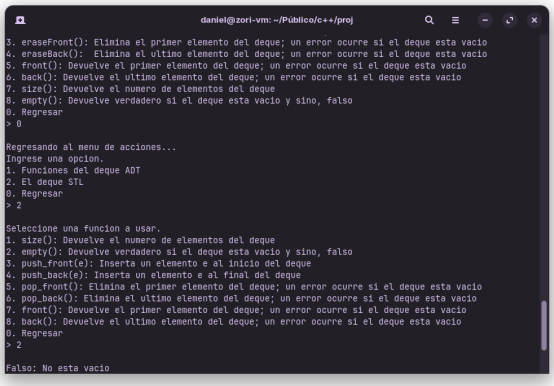


(c) Validación de elementos con *front()* y *back()*

Figura 2: (1/3) Insertamiento multiple de datos en el menú para deque ADT



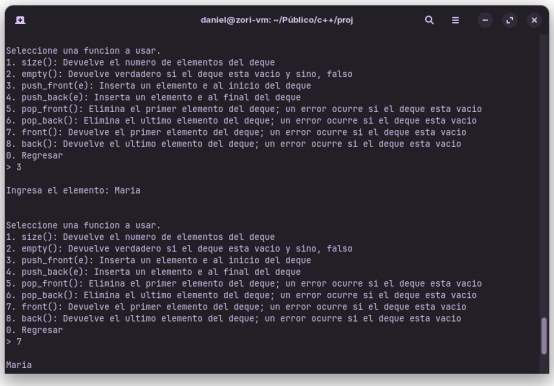
(a) Validación de elementos con *size()* y *empty()*



(b) Navegación para menú de deque STL

(c) Validación de elementos con *front()* y *back()* de deque STL

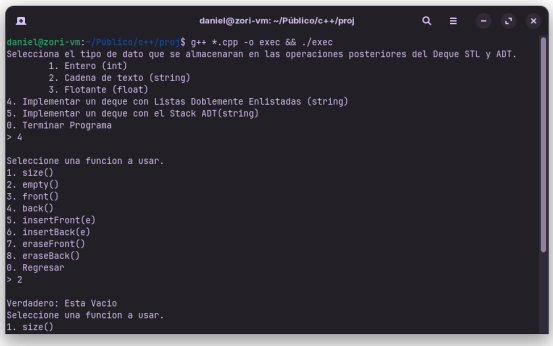
Figura 3: (2/3) Insertamiento multiple de datos en el menú para deque ADT y STL



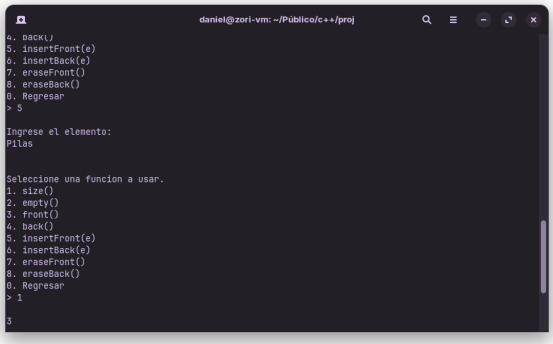
(a) Ingreso de elemento con *pushfront(e)*

(b) Validación con *front()*

Figura 4: (3/3) Insertamiento multiple de datos en el menú para deque STL

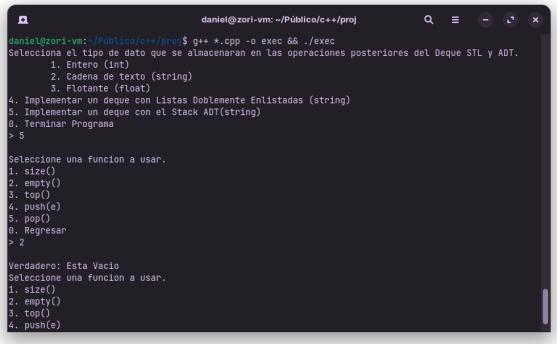


(a) Menú de Listas Doblemente Enlistadas



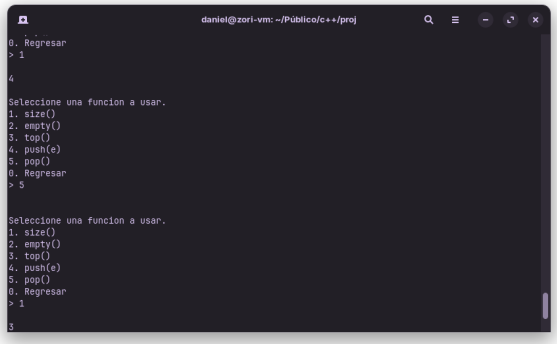
(b) Ingreso de datos (y previos) con *insertFront(e)* y validación de tamaño con *size*

Figura 5: Insertamiento multiple de datos en el menú para deque con Listas Doblemente Enlistadas



(a) Menú y validación con *empty()*

(b) Ingreso de elemento (y previos) con *push(e)* y validación con *size()*



(c) Eliminación con *pop()* e impresión de total de elementos con *size()*

Figura 6: Insertamiento multiple de datos en el menú para deque con el Stack ADT