Tarea 1

Estructuras de datos

Al ser que llevo un rato sin implementar estructuras de datos desde cero, decidí intentar desarrollar las estructuras por mi cuenta, en el caso de la cola y la pila, la implementación fue sencilla, solamente use los arreglos comunes de C++, tan solo defini arreglos de tamaño fijo para cada estructura, luego, para las pilas, cree una variable que representa al elemento mas nuevo de la pila y lo uso para acceder al elemento mas nuevo, luego hago las operaciones en torno a este indice, con la cola, agregué una variable mas para representar al ultimo elemento dentro de la fila y de esta manera tener acceso inmediato al elemento mas antiguo, posteriormente mejore mi acercamiento basándome en un tutorial con el fin de trabajar a estas estructuras de manera mas eficiente: https://medium.com/@RobuRishabh/understanding-how-to-use-stack-queues-c-9f1fc06d1c5e

En el caso de los mapas, tenia clara la teoria de como funciona el hasheo y el manejo de colisiones (por lista, o encontrando el espacio vacío mas cercano), al ser que se me complico un poco implementarlo, por eso me base en el siguiente tutorial que implementa una tabla con encadenación separada, lo que permite manejar colisiones de hash almacenando los valores con hash repetidos mediante listas enlazadas: https://medium.com/@omerhalidcinar/building-your-own-hashmap-in-c-open-addressing-separate-chaining-implementations-ead22ca955c2

Casos de Uso

Para las pilas y colas, se agregan 3 elementos iguales y luego se eliminan, el resultado debería de ser la misma fila de números impresa en orden contrario, en el caso de la pila seria 300, 200, y 100 por el orden de eliminación LIFO, y en caso de la cola seria 100, 200, y 300 por ser FIFO.

```
int main(int argc, const char * argv[]) {

    Stack stack(5);
    std::cout << "Stack" << std::endl;
```

```
    stack.push(100);
    stack.push(200);
    stack.push(300);
    std::cout << "Eliminacion de elementos: ";
    while (!stack.isEmpty()) {
        std::cout << stack.pop() << " " << std::endl;
    }
    std::cout << std::endl;

    Queue queue(5);
    std::cout << "Cola" << std::endl;
    queue.enqueue(100);
    queue.enqueue(200);
    queue.enqueue(300);
    std::cout << "Eliminacion de elementos ";
    while (!queue.isEmpty()) {
        std::cout << queue.dequeue() << " " << std::endl;
    }
    std::cout << std::endl;
```

Para el hash map, se almacenan distintos valores y luego se hace acceso a una llave y se actualiza.

```
    HashMap<std::string, int> scores;
    std::cout << "Hash Map" << std::endl;
    scores.insert("Alice", 95);
    scores.insert("Bob", 87);
    scores.insert("Charlie", 92);

    auto score = scores.get("Alice");

    if (score.has_value()) { // Comprobar que la llave tenga un valor
        std::cout << "Alice: " << score.value() << std::endl;
    }

    // ctualizar valor de Alice
    scores.insert("Alice", 98);
    score = scores.get("Alice");
    if (score.has_value()) {
        std::cout << "Se actualizo la calificacion de Alice: " <<
score.value() << std::endl;
    }
}
```

Link al repositorio: https://github.com/RosasSebastian2003/DSA-8.git

Referencias: