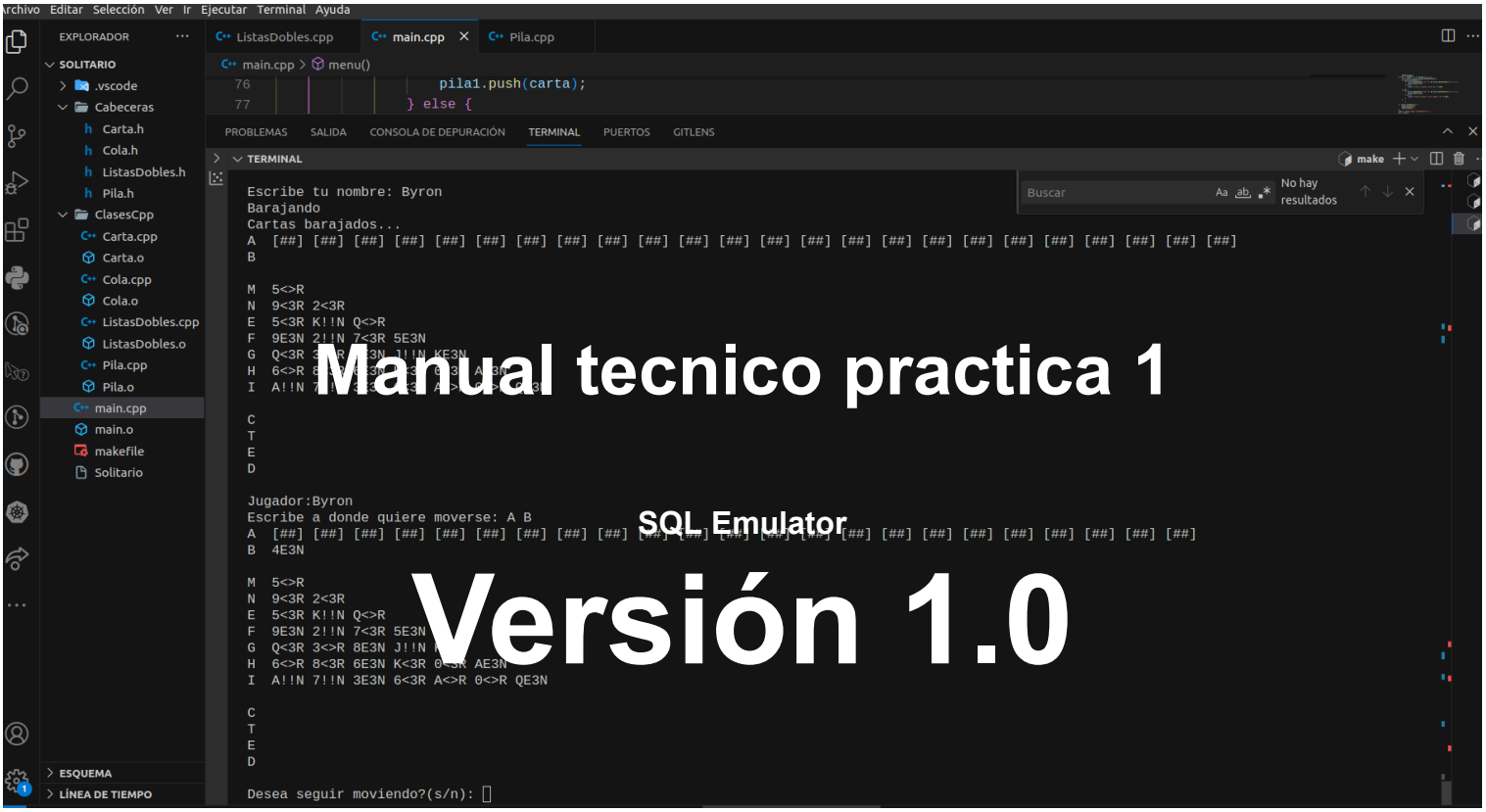
UNIVERSIDAD SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
DIVISIÓN DE CIENCIAS DE LA INGENIERÍA
Ingeniería en Ciencias y Sistemas.
Ing. Christian Quiroa
Aux: Yefer Alvarado

Estructura de datos

Byron Fernando Torres Ajxup 201731523



Manual tecnico practica 1

SQL Emulator

Versión 1.0

Quetzaltenango 08 de Marzo del 2024

Índice

Introducción

Las estructuras de datos son fundamentales en el desarrollo de software, ya que proporcionan un marco organizado y eficiente para almacenar, gestionar y acceder a datos. Estas estructuras forman la base sobre la cual se construyen algoritmos y

aplicaciones, influyendo directamente en la eficiencia, claridad y mantenibilidad del código.

Eficiencia en el Acceso y Manejo de Datos:

Las estructuras de datos bien diseñadas permiten un acceso rápido y eficiente a los datos, optimizando el rendimiento de las operaciones realizadas sobre ellos.
Algoritmos que utilizan estructuras de datos eficientes pueden reducir significativamente la complejidad temporal y mejorar la velocidad de ejecución.
Organización y Mantenibilidad del Código:

Utilizar estructuras de datos adecuadas facilita la organización del código, mejorando la claridad y comprensión del mismo.
La elección de la estructura de datos correcta contribuye a un código más limpio y fácil de mantener, ya que se adapta a la lógica y naturaleza de los datos manipulados.

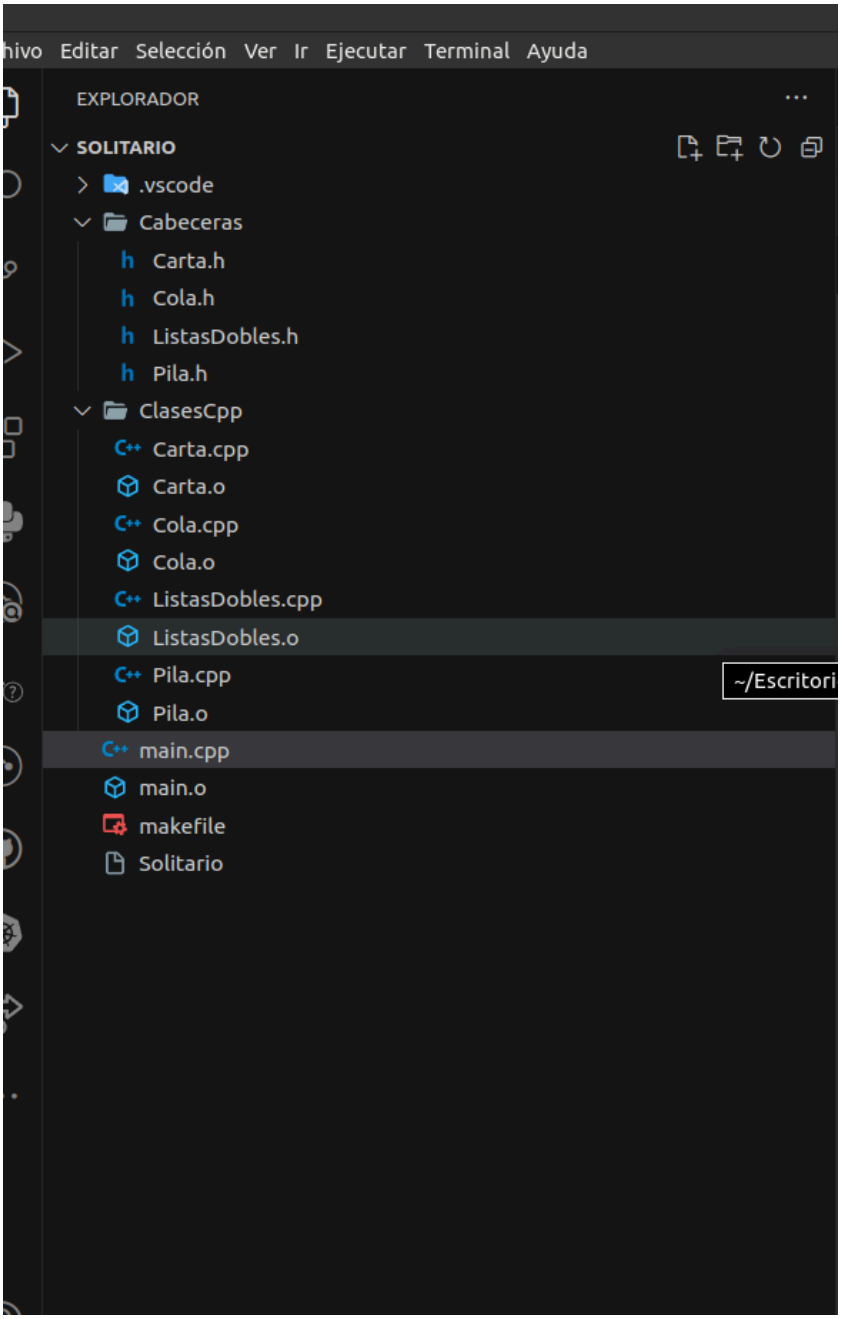
Reducción de la Complejidad Algorítmica:

Muchos problemas computacionales se pueden resolver de manera más efectiva mediante el uso de estructuras de datos apropiadas. Por ejemplo, el uso de un mapa o conjunto puede simplificar la búsqueda y manipulación de datos únicos.
Las estructuras de datos bien diseñadas pueden ayudar a reducir la complejidad de los algoritmos, haciendo que el código sea más sencillo y más fácil de entender.

Estructura del Proyecto:

carta.hpp y carta.cpp: Definen la clase Carta con atributos como color, palo y valor.
pila.hpp y pila.cpp: Definen la clase Pila que puede contener cartas y operaciones asociadas como push y pop.
cola.hpp y cola.cpp: Definen la clase Cola que puede contener cartas y operaciones asociadas como encolar y desencolar.
listaDobleEnlazada.hpp y listaDobleEnlazada.cpp: Definen la clase ListaDobleEnlazada que utiliza nodos para almacenar cartas y operaciones asociadas como inserción y eliminación.
nodo.hpp y nodo.cpp: Definen la estructura de nodo utilizada por la lista doblemente enlazada.

Árbol de carpetas



makefile

```
# QUI ESPECIFICO EL COMPILADOR
COMPILADOR = g++

# Bandera de compilación
BANDERA = -std=c++11 -Wall

# Directorios de los archivos de cabecera y de implementación
CARPETASCABECERA = Cabeceras
```

```
CARPETASCPP = ClasesCpp

# Obtén la lista de archivos de implementación
SRCS = $(wildcard $(CARPETASCPP)/*.cpp)

# Agrega el archivo main.cpp a la lista de archivos de implementación
SRCS += main.cpp

# Objetos generados a partir de los archivos de implementación
OBJS = $(SRCS:.cpp=.o)

# Nombre del ejecutable
NOMBRE = Solitario

# Regla para construir el ejecutable
$(NOMBRE): $(OBJS)
	$(COMPILADOR) $(BANDERA) -o $@ $^

# Regla para construir los objetos
%.o: %.cpp
	$(COMPILADOR) $(BANDERA) -I$(CARPETASCABECERA) -c -o $@ $<

# Regla para limpiar los archivos generados
clean:
	rm -f $(NOMBRE) $(OBJS)

# Regla para ejecutar el programa
run: $(NOMBRE)
	./$(NOMBRE)
```