

**República Bolivariana de Venezuela
Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Asignatura: Algoritmos y Programación**

INFORME DEL PROYECTO #2

Profesor

Ricardo Osuna

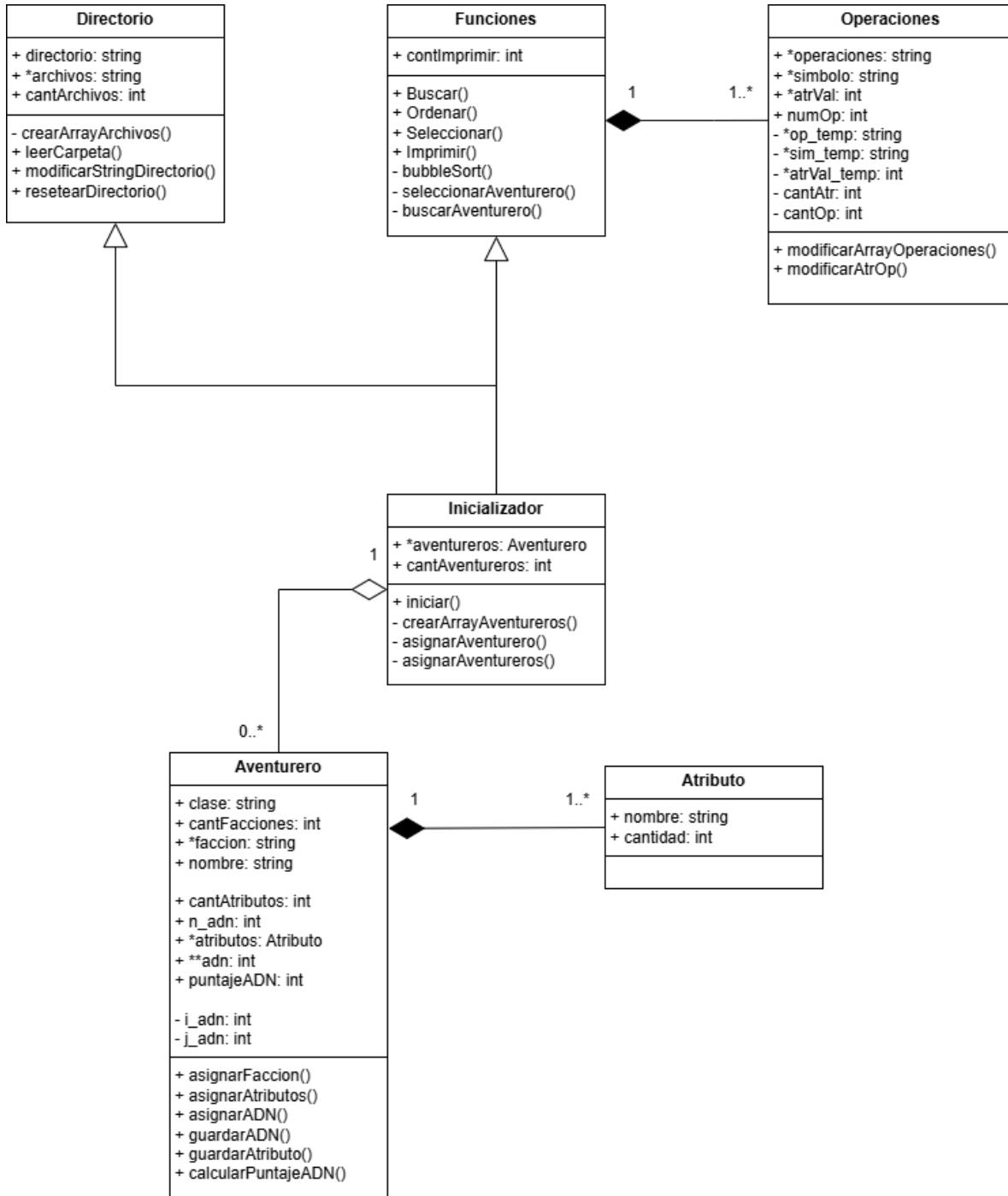
Estudiantes

José Maceiras
C.I. 31.228.863

Leonardo Espinola
C.I. 32.105.059

Caracas, 8 de marzo de 2025

DIAGRAMA DE CLASES



NOTA A LOS PREPARADORES: Al final del documento, hay unos comentarios y sería importante para nosotros que los leyeran.

CLASE "AVENTURERO"

La clase aventurero guarda la información recogida de cada archivo de texto, y cuenta con las variables correspondientes a la "clase" el nombre y la facción del aventurero. También cuenta con distintas funciones algunas simplemente para asignar un arreglo dinámico como `asignarFaccion()`, `asignarAdn()`, `asignarAtributos()`, y otras como:

`guardarADN()`: que se encarga de ir guardando los valores de la matriz del archivo para posteriormente usarlos.

La función `calcularPuntajeADN()`: encuentra la mitad de la matriz y comprueba primero las diagonales tomando sus valores sin contar la mitad de la matriz excepto el caso de los verdes el cual es el único que toma el valor del medio, cada valor que va tomando lo va convirtiendo a -1 (esto es porque en la matriz nunca habrá un número negativo) para después poder encontrar los valores blancos simplemente recorriendo la matriz de forma normal y omitiendo todo lo que sea -1, después con todos los datos saca el valor con la fórmula dada " $\text{puntajeADN} = \text{mult} - \text{puntajeADN}$ " y retorna `puntajeADN`.

CLASE "FUNCIONES"

La clase de funciones se encarga principalmente de segmentar el código y hacer más fácil su lectura, ya que `Inicializar()` hereda de esta clase. Contiene las funciones principales del programa, que son las de ordenar, seleccionar, buscar e imprimir.

Las funciones `Buscar()`, `Ordenar()` y `Seleccionar()`, reciben N cantidad de instrucciones respectivamente hasta que el usuario introduzca la palabra "INICIAR", estas instrucciones se deben guardar en algún lado para ejecutarlas cuando eso ocurra, ya que no se pueden ejecutar las funciones en cada entrada antes de iniciar ya que eso significaba modificar el contenido del arreglo de aventureros. Para ello la solución fue crear una especie de buffer (sitio donde guardar temporalmente la información para después utilizarla), que sería la función de Operaciones, donde en cada uno de los métodos crea un objeto de Operaciones y ahí va guardando las instrucciones en sus respectivos arreglos.

Luego se itera según la cantidad de operaciones que hay y se llama a las funciones auxiliares de cada método (que serían `bubbleSort()`, `seleccionarAventurero()`, `buscarAventurero()`, respectivamente), con los parámetros como qué tipo de operación se está realizando, su símbolo, etc.

Estos métodos auxiliares, que están declarados como privados, recorren todo el arreglo de aventureros y retornan al final cuantos ha encontrado en esa operación, esto es especialmente útil ya que lo que el programa hace es mover los aventureros que coincidan con los parámetros de la operación al inicio del arreglo de aventureros, y los que no atrás, dependiendo de cuantos coincidan al final, se va a cambiar el valor de

cuantos aventureros hay en el arreglo, y cuando termine la operación, cambiará el tamaño del arreglo de aventureros al tamaño del nuevo (de esto se encarga Inicializador::crearArrayAventureros(), más adelante se explica).

El método Imprimir() de esta clase no tiene mayor misterio, crea un directorio llamado operaciones, y ahí solo genera los respectivos archivos de los resultados de las operaciones hasta el momento de que el usuario envía el comando, para eso se recorre el arreglo de aventureros, si el usuario quiere imprimir más de una vez, lo podrá hacer, y generará un archivo distinto (operacion1.out en el primer IMPRIMIR, operacion2.out en el segundo, y así respectivamente).

CLASE "OPERACIONES"

Esta clase es útil para crear objetos de la misma en las funciones del programa, ya que permite almacenar en un "buffer" las operaciones que el usuario está ingresando y luego recorrer todas esas operaciones para ejecutarlas cuando el usuario escriba el comando INICIAR.

modificarAtrOp() y modificarArrayOperaciones() son funciones que permiten iniciar los arreglos en donde se guarda la información de las operaciones y cambiar el tamaño del arreglo de manera dinámica sin perder el contenido que estaba en ellos anteriormente, funcionan de manera similar a Inicializador::crearArrayAventureros().

CLASE "DIRECTORIO"

Directorio, es una clase de la cual Inicializador() hereda. Se encarga de leer qué archivos están dentro de una carpeta y almacenar los nombres de cada uno en un arreglo y cuantos archivos son en total. Tiene los siguientes métodos: leerCarpeta(), modificarStringDirectorio(), resetearDirectorio() y crearArrayArchivos().

leerCarpeta(): Recibe desde el inicializador el directorio y ejecuta el comando de windows "dir /b "directorio" > database.temp" para recuperar todos los archivos de X directorio y guardar el resultado del comando en un archivo, luego, se recorre cada línea de ese archivo para saber cuantos archivos hay, y posteriormente crear el arreglo de archivos con crearArrayArchivos(cantidad de archivos), luego, se recorre nuevamente el archivo temporal generado pero esta vez asignando el contenido de cada línea a ese arreglo (el arreglo se usa para poder leer cada archivo de aventurero y guardar los datos de cada uno en el arreglo de aventureros), finalmente se borra el archivo temporal.

modificarStringDirectorio(): se asegura de que el formato del directorio actual que envió el usuario en la función de cargar sea el correcto, se encarga de quitar comillas innecesarias y de poner un "/" al final siempre.

resetearDirectorio(): devuelve a los valores iniciales el directorio y borra el arreglo con los nombres de los archivos, útil para poder llamar a la función CARGAR varias veces. Se llama después de haber cargado a todos los aventureros.

CLASE "INICIALIZADOR"

`crearArraydeAventureros()`: Crea el arreglo de aventureros y en caso de que hayan nuevos aventureros se le suman al arreglo o en caso de que en las funciones de búsqueda se filtre disminuya el tamaño del arreglo.

Funcion `asignarAventureros()`: Esta funcion es la que lee directamente el archivo del aventurero, a su vez que lo va leyendo la primera vez se esta escribiendo otro archivo llamado `archivo_manipulado.txt` el cual es el mismo archivo pero reemplazando cualquier ":" por " " esto para poder leer el archivo palabra por palabra, todo esto a su vez que encontramos la posicion en la que estan el ADN y la los atributos, terminado esto recorremos el `archivo_manipulado.txt`, este segundo recorrido lo que hace es comprobar si la palabra es igual a Clase Faccion o nombre, para nombre y clase lo que se hace es un `getline` que te da toda la linea despues de Nombre o Clase y para Faccion es exactamente lo mismo pero este al poder ser distintas facciones lo manejamos como un arreglo al que pertenece, entonces lo que se hace es recorrer la linea para encontrar la cantidad de "[" que existan ya que esa +1 es la cantidad de facciones que hay en un aventurero, una vez termina eso, se crea el arreglo de facciones y se hace un doble ciclo que recorra la linea caracter por caracter y vaya creando en la posicion "i" del arreglo la faccion correspondiente hasta que consiga un "]" en ese caso se detiene el segundo ciclo, se suma una iteracion al primero y se continua y asi sacamos todas las facciones despues se llama a una funcion que quita los espacios en blanco de una cadena de texto y se asigna el arreglo de facciones al aventurero: Para conseguir los atributos hay un condicional que pregunta la posición del ADN y la de los Atributos anteriormente encontrados y mientras "i" esté dentro de ese rango tomara todo como atributos, en caso de "i" ser mayor que `posADN` empezara a guardar el ADN dentro del aventurero iterado.

`Iniciar()`: es la función en la que ingresas los valores con un ciclo `while` y dependiendo de la instrucción se llama a la función correspondiente

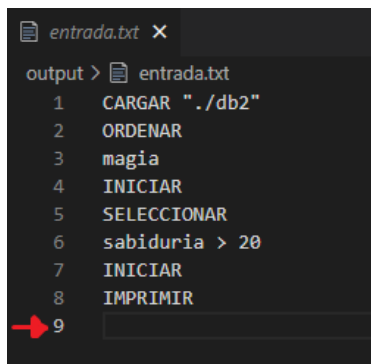
NOTA A LOS PREPARADORES:

El proyecto lo hemos probado y ha dado muy buenos resultados, pero hay un pequeño problema a la hora de usar un archivo para pasar los valores a la entrada del programa (solo ocurre ahí), y sabemos que hasta donde sabemos ustedes usan ese método (ej: "output.exe < entrada.txt") para facilitar la corrección, es más, nosotros también hacemos eso para ir probando el programa, si no, sería tedioso introducir los mismos datos una y otra vez.

El problema es el siguiente, nosotros tenemos el programa constantemente ejecutado en un ciclo while, entonces, si le pasamos un archivo tuvimos un problema inicial que consistía en que se pasaban los datos no una vez, si no en un bucle infinito. Esto lo logramos solucionar con esta línea de código:

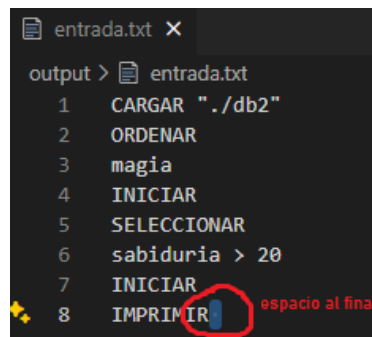
```
cin >> dato;  
if (cin.eof()) break; // Previene un bucle infinito si se lee la entrada de un archivo.  
if(dato == "CARGAR") {
```

Ahora no hay problema con ese bucle. Pero... si no hay un espacio o un enter al final del archivo de prueba, no lee la última instrucción.



```
output > entrada.txt  
1 CARGAR './db2'  
2 ORDENAR  
3 magia  
4 INICIAR  
5 SELECCIONAR  
6 sabiduria > 20  
7 INICIAR  
8 IMPRIMIR  
9
```

(enter al final)



```
output > entrada.txt  
1 CARGAR './db2'  
2 ORDENAR  
3 magia  
4 INICIAR  
5 SELECCIONAR  
6 sabiduria > 20  
7 INICIAR  
8 IMPRIMIR
```

Estos son los casos en los que si lee correctamente las instrucciones. Si no hubiera alguno de esos casos dos, omitiría la última instrucción que es IMPRIMIR (en ese ejemplo)

Estuvimos investigando y eso ocurre debido a que si no hubiera ese espacio/enter, el programa toma el IMPRIMIR como el EOF (final del archivo). Entonces, les pedimos que tengan en consideración eso a la hora de la corrección para que no nos perjudique, ya que el programa como tal funciona a la perfección tanto si introduces los datos manualmente, como de esta manera, solo que nos ocurrió esto que nos generó una preocupación, ya que como tal no es algo que forma parte del problema que propone el proyecto.