

UNIVERSIDAD DE ALCALÁ
Ingeniería en Sistemas de Información

Estructuras de Datos
Curso 2020-2021

PECL1 (SIMULACIÓN DEL CONTROL DE ACCESO A UN EVENTO DEPORTIVO)

ADINA MURG - [REDACTED]

VICTORIA LORENA ORDENES ORBEGOZO - [REDACTED]



ÍNDICE DE CONTENIDO:

1. Detalles y justificación de la implementación
 - 1.1 Aficionado
 - 1.2 Gestor
 - 1.3 Pila
 - 1.4 Nodo Pila
 - 1.5 Cola
 - 1.6 Nodo Cola
 - 1.7 Lista
 - 1.8 Nodo Lista
2. Solución adoptada: Descripción de Dificultades Encontradas.
3. Diagrama UML
4. Explicación de los métodos métodos más destacados
5. Explicación del comportamiento del programa
6. Bibliografía

1. Detalles y justificación de la implementación

En esta sección comentaremos los TAD's que se han empleado a lo largo del programa, su especificación y las operaciones que emplean.

1.1 Aficionado

espec AFICIONADO

usa NATURAL, BOOL

generos aficionado

operaciones

esSocio: aficionado-> bool

mostrar: aficionado-> aficionado

getId: aficionado -> natural

getMllegada: aficionado -> natural

var

id, mllegada: natural

socio: bool

1.2 Gestor

espec GESTOR

usa PILA, COLA, LISTA, AFICIONADO

generos gestor

operaciones

genera10Aficionados:-> pila

borrarAficionados: pila-> pila

muestraAficionados: pila-> aficionado

encolarAficionados: pila -> cola

muestraSocios: cola -> aficionado

muestraSimpatizantes: cola -> aficionado

vacíaColas: cola-> cola

muestraAficionadosEnLista: cola cola -> lista

buscaSocios: lista -> aficionado

reiniciar: lista -> lista

pila-> pila

cola -> cola

var

colaSocios, colaSimpatizantes: cola

listaSocios: lista

pilaAficionados: pila

1.3 Pila

espec PILA

usa AFICIONADO, BOOL

generos pila

operaciones

apilarA: pila, aficionado-> pila

mostrarPila: pila-> aficionado

pilavacia: pila-> bool

desapilar: pila -> aficionado

var

cima: nodo pila

longitud: natural

1.4 Nodo Pila

espec NODOPILA

usa NATURAL, BOOL

generos pnode

1.5 Cola

espec COLA

usa AFICIONADO, BOOL, NATURAL

generos cola

operaciones

encolar: cola, aficionado-> cola

colaVacía: cola> bool

mostrarCola: cola> aficionado

desencolar: cola-> aficionado

getLongitud: cola-> natural

var

primero, ultimo: nodo cola

longitud: natural

1.6 Nodo Cola

espec NODOCOLA

usa NATURAL, BOOL

generos cnodo

1.7 Lista

espec LISTA

usa AFICIONADO, NATURAL

generos lista

operaciones

insertarOrden: aficionado-> lista

listaVacía: lista-> lista

mostrarLista: lista-> aficionado

concatenar: lista lista-> lista

getLongitud: lista-> natural

elemento: lista -> aficionado

var

primero: nodo lista

longitud: natural

1.8 Nodo Lista

espec NODOLISTA

usa NATURAL, BOOL

generos lnodo

2. Solución adoptada: Descripción de Dificultades Encontradas

La **primera dificultad** con la que nos encontramos fue la de conseguir generar el Id del pasajero de forma aleatoria y sin repetición.

Planteamos muchos métodos para resolver dicho problema, pero ninguno de ellos era óptimo. Para resolverlo, con la ayuda de las tutorías llegamos a la conclusión de usar el método shuffle, que gracias a un array de 9 números y tomando en consideración la longitud de la pila, generamos ese Id que irá aumentando progresivamente lo haga la pila y sin generar repeticiones.

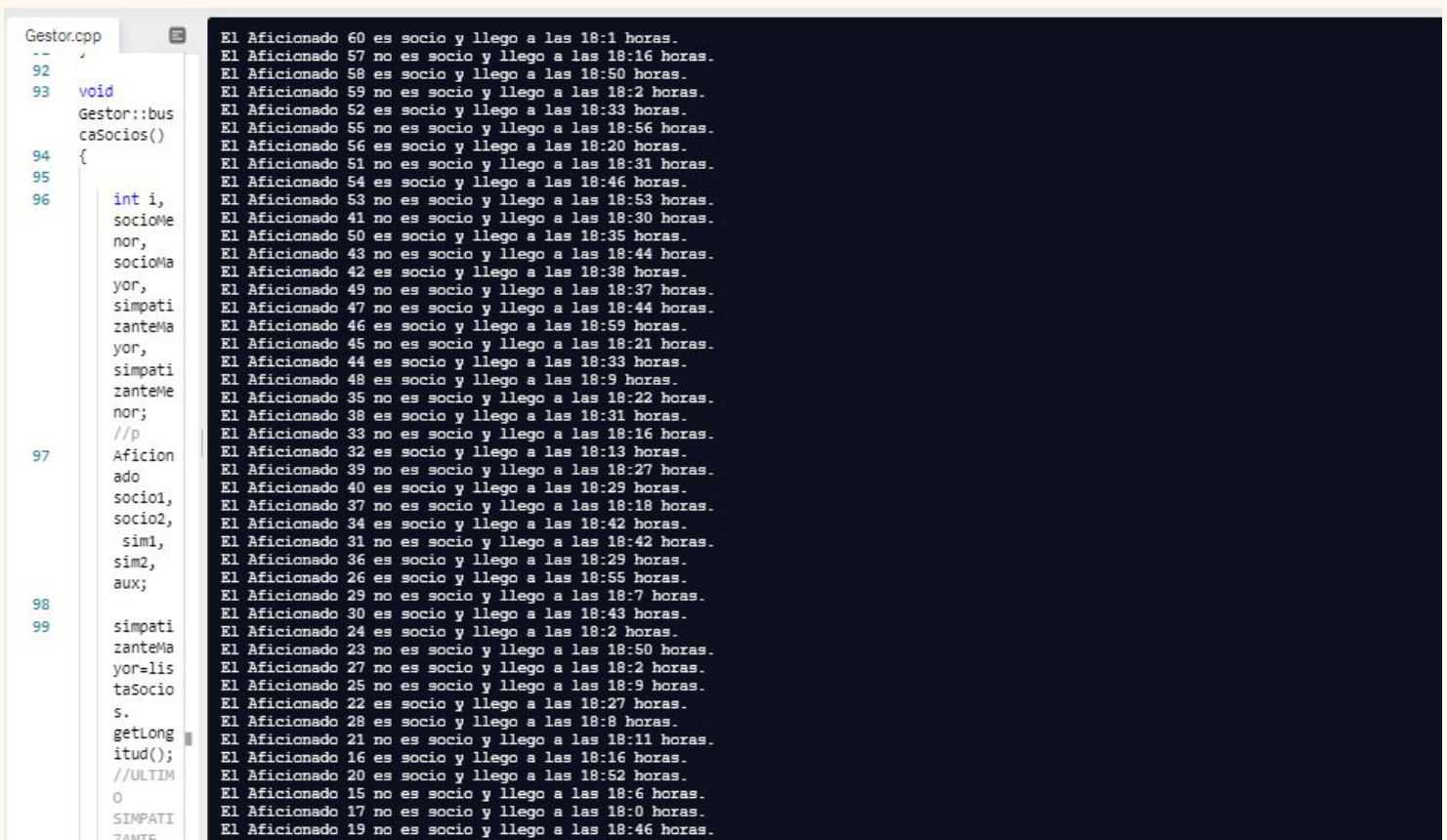
La **segunda dificultad**, fue la de organizar/comprender cómo se organizaban las clases y cómo podíamos acceder a ellas. Ejemplo, en la clase gestor: `pilaAficionados.getLongitud()`

con la que podemos acceder al atributo longitud de pila. La solución la descubrimos mediante prueba-error.

La **tercera dificultad** fue el manejo de punteros, aunque no fue muy considerable debido a que ya los habíamos tratado en ejercicios teóricos, y gracias a comprenderlos, los pudimos aplicar y adaptar en nuestro código, como por ejemplo, el método para insertar en orden de la clase lista. Aplicando dichos conocimientos y aplicando la lógica, resolvimos muchos otros, como por ejemplo el método de concatenar listas.

La **cuarta dificultad**, aunque más que dificultad llamaremos inconveniencia, fue el manejo de CodeLite, dado que el programa y el compilador gestionaban mal el programa. Por ejemplo, al insertar 4 ceros, nos debería generar 40 aficionados al pulsar el botón 1. Pero esto no sucede así, debemos pulsar el botón 1 más de una vez para que CodeLite refresque bien la información. Para asegurar que esto no era un problema nuestro, ejecutamos el código en un compilador online (<https://repl.it/>) y observamos en que este, indiferentemente del número de 0 que introducimos y el número de aficionados que se generen, si pulsamos el botón el 1 los carga bien y sin problema.

En esta imagen podemos observar que en repositorio online se cargan los valores sin problema.



```
Gestor.cpp
--
92
93 void
Gestor::bus
caSocios()
94 {
95
96     int i,
sociome
nor,
socioma
yor,
simpati
zanteMa
yor,
simpati
zanteMe
nor;
//p
97 Aficion
ado
socios1,
socios2,
sim1,
sim2,
aux;
98
99 simpati
zanteMa
yor=lis
taSocio
s.
getLong
itud();
//ULTIM
O
SIMPATI
ZANTE
El Aficionado 60 es socio y llego a las 18:1 horas.
El Aficionado 57 no es socio y llego a las 18:16 horas.
El Aficionado 58 es socio y llego a las 18:50 horas.
El Aficionado 59 no es socio y llego a las 18:2 horas.
El Aficionado 52 es socio y llego a las 18:33 horas.
El Aficionado 55 no es socio y llego a las 18:56 horas.
El Aficionado 56 es socio y llego a las 18:20 horas.
El Aficionado 51 no es socio y llego a las 18:31 horas.
El Aficionado 54 es socio y llego a las 18:46 horas.
El Aficionado 53 no es socio y llego a las 18:53 horas.
El Aficionado 41 no es socio y llego a las 18:30 horas.
El Aficionado 50 es socio y llego a las 18:35 horas.
El Aficionado 43 no es socio y llego a las 18:44 horas.
El Aficionado 42 es socio y llego a las 18:38 horas.
El Aficionado 49 no es socio y llego a las 18:37 horas.
El Aficionado 47 no es socio y llego a las 18:44 horas.
El Aficionado 46 es socio y llego a las 18:59 horas.
El Aficionado 45 no es socio y llego a las 18:21 horas.
El Aficionado 44 es socio y llego a las 18:33 horas.
El Aficionado 48 es socio y llego a las 18:9 horas.
El Aficionado 35 no es socio y llego a las 18:22 horas.
El Aficionado 38 es socio y llego a las 18:31 horas.
El Aficionado 33 no es socio y llego a las 18:16 horas.
El Aficionado 32 es socio y llego a las 18:13 horas.
El Aficionado 39 no es socio y llego a las 18:27 horas.
El Aficionado 40 es socio y llego a las 18:29 horas.
El Aficionado 37 no es socio y llego a las 18:18 horas.
El Aficionado 34 es socio y llego a las 18:42 horas.
El Aficionado 31 no es socio y llego a las 18:42 horas.
El Aficionado 36 es socio y llego a las 18:29 horas.
El Aficionado 26 es socio y llego a las 18:55 horas.
El Aficionado 29 no es socio y llego a las 18:7 horas.
El Aficionado 30 es socio y llego a las 18:43 horas.
El Aficionado 24 es socio y llego a las 18:2 horas.
El Aficionado 23 no es socio y llego a las 18:50 horas.
El Aficionado 27 no es socio y llego a las 18:2 horas.
El Aficionado 25 no es socio y llego a las 18:9 horas.
El Aficionado 22 es socio y llego a las 18:27 horas.
El Aficionado 28 es socio y llego a las 18:8 horas.
El Aficionado 21 no es socio y llego a las 18:11 horas.
El Aficionado 16 es socio y llego a las 18:16 horas.
El Aficionado 20 es socio y llego a las 18:52 horas.
El Aficionado 15 no es socio y llego a las 18:6 horas.
El Aficionado 17 no es socio y llego a las 18:0 horas.
El Aficionado 19 no es socio y llego a las 18:46 horas.
```

Nuestro problema:

Consideramos que nuestro problema funciona de forma correcta y eficiente hasta cierto punto.

El problema empieza en el punto 9, con reiniciar. Desde este punto, llamamos a los destructores de pila, cola y lista. Con cola y lista el programa funciona de forma correcta y eficiente, pero el problema empieza con el destructor de pila, dado que al reiniciar listas, el programa se queda atascado, debido al error *Segmentation Fault*.

Este error como comentamos, nos genera problemas con el punto 7 y 8 al reiniciar. Tratamos de corregir el problema, mediante un if-else, como el que se adjunta en la imagen para el apartado 7 y 8. Si aplicamos estos if-else, al reiniciar, el programa funcionará de forma correcta e imprimirá el mensaje de “La lista esta vacia”.

Ahora, el problema de hacer esto, nos genera otro, y es que debido a lo que hemos comentado anteriormente, debemos pulsar más de una vez el botón para que CodeLite refresque bien la información, y debido a esto debemos pulsar varias veces el botón. Dado que debemos pulsar varias veces el 7 para que cargue bien los aficionados, el programa mostrará el mensaje de “La lista esta vacia” al pulsar el mismo botón 2 veces, ya que al haber desencolado para mostrar la primera lista no habrá más elementos y mostrará dicho mensaje.

```
88 void Gestor::buscaSocios()
89 {
90     int i,socioMenor,socioMayor,simpatizanteMayor,simpatizanteMenor; //Posiciones.
91     Aficionado socio1,socio2, sim1,sim2,aux;
92
93     simpatizanteMayor=listaSocios.getLongitud();//Último simpatizante de la lista.
94     socioMayor=(simpatizanteMayor)/2;// Último socio de la lista.
95     simpatizanteMenor=(simpatizanteMayor/2)+1;// Primer simpatizante de la lista.
96     socioMenor=1;// Primer socio de la lista.
97     i=1;
98
99     if(listaSocios.getLongitud()>0){
100         while(listaSocios.getLongitud()>0){
101             if(i==socioMenor){
102                 socio1=listaSocios.elemento();
103             }else if(i==socioMayor){
104                 socio2=listaSocios.elemento();
105             }else if(i==simpatizanteMenor){
106                 sim1=listaSocios.elemento();
107             }else if(i==simpatizanteMayor){
108                 sim2=listaSocios.elemento();
109             }else{
110                 aux=listaSocios.elemento();
111             }
112             i++;
113         }
114         cout<<"El primer socio es :",socio1.mostrar();
115         cout<<"El ultimo socio es :",socio2.mostrar();
116         cout<<"El primer simpatizante es :",sim1.mostrar();
117         cout<<"El ultimo simpatizante es :",sim2.mostrar();
118     }else{
119         cout<<"La lista esta vacia.";
120     }
121 }
122
```



```

72 void Gestor::muestraAficionadosEnLista(){
73
74     Lista listaSimpatizantes; // Lista auxiliar
75     if(colaSocios.getLongitud()==0){
76         cout<<"La lista esta vacia.";
77     }else{
78
79         while(colaSocios.getLongitud()>0){
80             Aficionado af1=colaSocios.desencolar();
81             listaSocios.insertarOrden(af1);
82         }
83         while(colaSimpatizantes.getLongitud()>0){
84             Aficionado af2=colaSimpatizantes.desencolar();
85             listaSimpatizantes.insertarOrden(af2); //Almacenamos en esta lista auxiliar los aficionados simpatizantes concatenados.
86         }
87         listaSocios.concatenar(listaSimpatizantes);
88         listaSocios.mostrarLista();
89     }
90 }

```

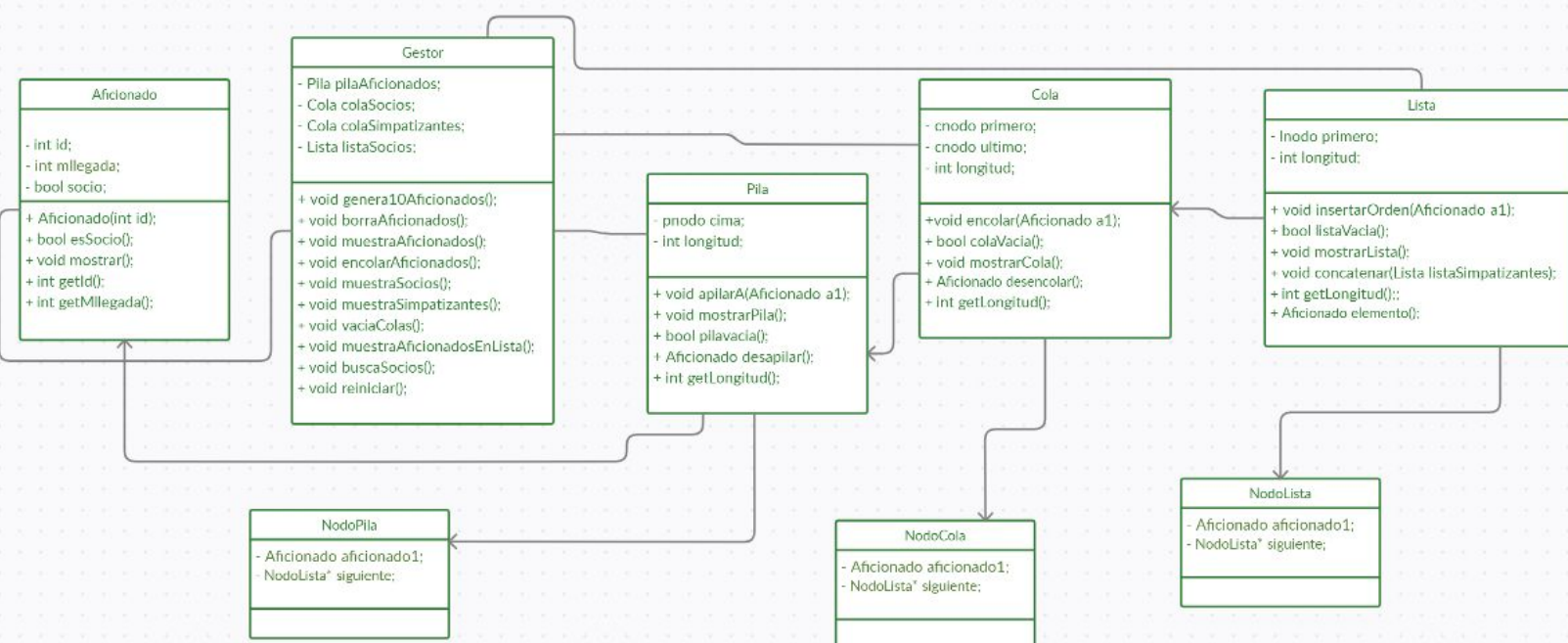
LINK A LA VERSIÓN ALTERATIVA:

<https://repl.it/@ijiouhj/PECL1#Gestor.cpp>

3. Diagrama UML

Diagrama UML. Las clases Nodo Cola son accesibles únicamente desde sus respectivos TAD's. La clase Aficionado es accesible para Pila, que a su vez es accesible para Cola y ésta lo es para Lista.

La clase Aficionado se relaciona con Gestor, que mantiene asociaciones con Pila, Cola y Lista.



4. Explicación de los métodos más destacados

Estos son los métodos que consideramos más destacados, ya que son de los más relevantes y que más dificultad nos ha costado. Algunos métodos, como mostrarLista, mostrarPila y mostrarCola, se asemejan entre ellos y por ese motivo hemos decidido no incluirlos. Otros métodos, como desencolar o desapilar, son idénticos a los vistos en teoría por lo que también los consideramos no relevantes.

Clase	Nombre	Argumentos	Retorno	Descripción
Gestor	generar10aficionados	--	Pila	Este método genera 10 aficionados cada uno con su respectivo id y los almacena en una pila.
Gestor	buscaSocios	Lista	Aficionado	Este método buscará en la lista que almacena todos los aficionados, a los primeros socios y simpatizantes en llegar. También a los últimos socios y simpatizantes que hay.
Lista	concatenar	Lista Lista	Lista	Este método es utilizado dentro de otro método del que hablaremos más adelante. Nos permite concatenar la lista de simpatizantes ordenada con la lista de socios en una única lista que es la que nos piden por requisitos del programa.
Lista	insertarOrden	Aficionado	Lista	Este método es el que nos permite insertar en una lista los aficionados de menor a mayor llegada.
Cola	desencolar	Cola	Aficionado	Nuestro método elimina al primer aficionado de la cola pero luego devuelve el objeto.

Pila	desapilar	Pila	Aficionado	Nuestro método elimina la cima de la pila pero luego devuelve el objeto que hay que es un aficionado.
Gestor	mostrarAficionadosEn Lista	Cola Cola	Lista	Este método se servirá de una lista auxiliar en memoria local donde almacenaremos a los simpatizantes ordenados que luego concatenamos para generar la lista única con todos los aficionados.
Lista	elemento	Lista	Aficionado	Este método elimina un aficionado de la lista pero ese aficionado nos lo devolverá ya que lo utilizamos.

5. Explicación del comportamiento del programa

El programa que hemos realizado tiene 9 opciones, lo manejamos a través de llamadas a métodos del objeto gestor que declaramos al principio en el main.cpp.

Opción 0 :

En esta opción cada vez que pulsemos se generarán 10 aficionados de forma aleatoria con un id único que almacenaremos en una pila.

Opción 1 :

En esta opción mostrará los elementos de la pila que hemos generado en opción 0. Los aficionados mostrarán su id, si es socio o no y a la hora en la que llegó.

```
El Aficionado 2 es socio y llego a las 18:44 horas.
El Aficionado 4 es socio y llego a las 18:22 horas.
El Aficionado 6 es socio y llego a las 18:18 horas.
El Aficionado 10 es socio y llego a las 18:18 horas.
El Aficionado 5 no es socio y llego a las 18:4 horas.
El Aficionado 1 no es socio y llego a las 18:29 horas.
El Aficionado 7 no es socio y llego a las 18:40 horas.
El Aficionado 3 no es socio y llego a las 18:34 horas.
El Aficionado 9 no es socio y llego a las 18:47 horas.
El Aficionado 8 es socio y llego a las 18:41 horas.
```

```
-----
0 Aficionados en la pila. 0 Socios en cola. 0 Simpatizantes en cola.
-----
```

0. Generar 10 aficionados de forma aleatoria.
1. Consultar todos los aficionados generados en la Pila (pendientes de entrar en las colas).
2. Borrar los aficionados generados en la pila.
3. Simular llegada de los aficionados en las colas.
4. Consultar la cola de aficionados socios.
5. Consultar la cola de aficionados no socios.
6. Borrar los todos los aficionados de las colas.
7. Simular la entrada de los aficionados al estadio (a la lista).
8. Buscar en la lista el socio y el simpatizante que m|is temprano y m|is tarde han llegado al estadio.
9. Reiniciar el programa.
- S. Salir.

Indique la opcion deseada:

Opción 2 :

En esta opción borramos los aficionados de la pila y nos aparecerá el menú para seleccionar otras opciones.

Opción 3 :

Aquí simularemos la salida de los aficionados de la pila y conforme los vayamos sacando irán a la cola de socios o simpatizantes, en función de su id es par o impar.

```
Vamos a encolar los aficionados a las colas
```

```
-----
0 Aficionados en la pila. 0 Socios en cola. 0 Simpatizantes en cola.
-----
```

0. Generar 10 aficionados de forma aleatoria.
1. Consultar todos los aficionados generados en la Pila (pendientes de entrar en las colas).
2. Borrar los aficionados generados en la pila.
3. Simular llegada de los aficionados en las colas.
4. Consultar la cola de aficionados socios.
5. Consultar la cola de aficionados no socios.
6. Borrar los todos los aficionados de las colas.
7. Simular la entrada de los aficionados al estadio (a la lista).
8. Buscar en la lista el socio y el simpatizante que m|is temprano y m|is tarde han llegado al estadio.
9. Reiniciar el programa.
- S. Salir.

Indique la opcion deseada:

Opción 4:

Esta opción mostrará los aficionados que son socios.

```
El Aficionado 8 es socio y llego a las 18:27 horas.
El Aficionado 2 es socio y llego a las 18:2 horas.
El Aficionado 6 es socio y llego a las 18:11 horas.
El Aficionado 4 es socio y llego a las 18:1 horas.
El Aficionado 10 es socio y llego a las 18:1 horas.

-----
0 Aficionados en la pila. 0 Socios en cola. 0 Simpatizantes en cola.
-----

0. Generar 10 aficionados de forma aleatoria.
1. Consultar todos los aficionados generados en la Pila (pendientes de entrar en las colas).
2. Borrar los aficionados generados en la pila.
3. Simular llegada de los aficionados en las colas.
4. Consultar la cola de aficionados socios.
5. Consultar la cola de aficionados no socios.
6. Borrar los todos los aficionados de las colas.
7. Simular la entrada de los aficionados al estadio (a la lista).
8. Buscar en la lista el socio y el simpatizante que m|is temprano y m|is tarde han llegado al estadio.
9. Reiniciar el programa.
S. Salir.

Indique la opcion deseada:
```

Opción 5:

Esta opción mostrará los aficionados que son simpatizantes:

```
El Aficionado 9 no es socio y llego a las 18:36 horas.
El Aficionado 5 no es socio y llego a las 18:55 horas.
El Aficionado 7 no es socio y llego a las 18:27 horas.
El Aficionado 3 no es socio y llego a las 18:5 horas.
El Aficionado 1 no es socio y llego a las 18:5 horas.

-----
0 Aficionados en la pila. 0 Socios en cola. 0 Simpatizantes en cola.
-----

0. Generar 10 aficionados de forma aleatoria.
1. Consultar todos los aficionados generados en la Pila (pendientes de entrar en las colas).
2. Borrar los aficionados generados en la pila.
3. Simular llegada de los aficionados en las colas.
4. Consultar la cola de aficionados socios.
5. Consultar la cola de aficionados no socios.
6. Borrar los todos los aficionados de las colas.
7. Simular la entrada de los aficionados al estadio (a la lista).
8. Buscar en la lista el socio y el simpatizante que m|is temprano y m|is tarde han llegado al estadio.
9. Reiniciar el programa.
S. Salir.

Indique la opcion deseada:
```

Opción 6:

En esta opción borraremos los aficionados de ambas colas.

Opción 7:

En esta opción simularemos la entrada de los aficionados que están en las colas a la lista de forma ordenada por el minuto de llegada. Tendrán prioridad los aficionados de la cola socios y después los simpatizantes.

```
El Aficionado 2 es socio y llego a las 18:2 horas.
El Aficionado 6 es socio y llego a las 18:22 horas.
El Aficionado 10 es socio y llego a las 18:33 horas.
El Aficionado 4 es socio y llego a las 18:38 horas.
El Aficionado 3 no es socio y llego a las 18:21 horas.
El Aficionado 7 no es socio y llego a las 18:35 horas.
El Aficionado 9 no es socio y llego a las 18:52 horas.
El Aficionado 1 no es socio y llego a las 18:56 horas.

-----
0 Aficionados en la pila. 0 Socios en cola. 0 Simpatizantes en cola.
-----

0. Generar 10 aficionados de forma aleatoria.
1. Consultar todos los aficionados generados en la Pila (pendientes de entrar en las colas).
2. Borrar los aficionados generados en la pila.
3. Simular llegada de los aficionados en las colas.
4. Consultar la cola de aficionados socios.
5. Consultar la cola de aficionados no socios.
6. Borrar los todos los aficionados de las colas.
7. Simular la entrada de los aficionados al estadio (a la lista).
8. Buscar en la lista el socio y el simpatizante que m|ís temprano y m|ís tarde han llegado al estadio.
9. Reiniciar el programa.
S. Salir.

Indique la opcion deseada:
```


Opción 8 :

En esta opción ya tenemos socios y simpatizantes en la lista. Queremos que aparezcan los primeros y últimos socios y simpatizantes.

```
El primer socio es :El Aficionado 2 es socio y llego a las 18:2 horas.
El ultimo socio es :El Aficionado 4 es socio y llego a las 18:38 horas.
El primer simpatizante es :El Aficionado 3 no es socio y llego a las 18:21 horas.
El ultimo simpatizante es :El Aficionado 1 no es socio y llego a las 18:56 horas.

-----
0 Aficionados en la pila. 0 Socios en cola. 0 Simpatizantes en cola.
-----

0. Generar 10 aficionados de forma aleatoria.
1. Consultar todos los aficionados generados en la Pila (pendientes de entrar en las colas).
2. Borrar los aficionados generados en la pila.
3. Simular llegada de los aficionados en las colas.
4. Consultar la cola de aficionados socios.
5. Consultar la cola de aficionados no socios.
6. Borrar los todos los aficionados de las colas.
7. Simular la entrada de los aficionados al estadio (a la lista).
8. Buscar en la lista el socio y el simpatizante que m|is temprano y m|is tarde han llegado al estadio.
9. Reiniciar el programa.
S. Salir.

Indique la opcion deseada:
```

Opción 9 :

Aquí reiniciaremos todas las estructuras de datos del programa para así poder volver a las demás opciones.

```
Vamos a reiniciar
-----
0 Aficionados en la pila. 0 Socios en cola. 0 Simpatizantes en cola.
-----

0. Generar 10 aficionados de forma aleatoria.
1. Consultar todos los aficionados generados en la Pila (pendientes de entrar en las colas).
2. Borrar los aficionados generados en la pila.
3. Simular llegada de los aficionados en las colas.
4. Consultar la cola de aficionados socios.
5. Consultar la cola de aficionados no socios.
6. Borrar los todos los aficionados de las colas.
7. Simular la entrada de los aficionados al estadio (a la lista).
8. Buscar en la lista el socio y el simpatizante que m|is temprano y m|is tarde han llegado al estadio.
9. Reiniciar el programa.
S. Salir.

Indique la opcion deseada:
```

Opción S:

En esta opción salimos del sistema.

6. Bibliografía

Apuntes, diapositivas, ejercicios y tutorías de clase:

BlackBoard - Estructuras de Datos.

Para generar los números aleatorios y sin repetir del id:

<https://www.geeksforgeeks.org/shuffle-an-array-using-stl-in-c/>

Para realizar el diagrama UML:

<https://www.lucidchart.com/pages/es/tutorial-de-diagrama-de-clases-uml>

Para corregir y entender errores:

<https://www.tutorialspoint.com/List-of-Common-Reasons-for-Segmentation-Faults-in-C-Cplusplus>

Herramienta para crear el diagrama UML:

<https://creately.com/es/lp/herramienta-de-diagrama-uml/>

Herramienta para trabajar en línea y compartido el proyecto de C++:

<https://repl.it/>