```
/*
 * File:   main.cpp
 * Author: ANA RONCAL
 * Created on 31 de agosto de 2023, 11:13 PM
 */

#include <cstdlib>
#include "Lista.h"
using namespace std;
#include "funcionesLista.h"
/*
* LISTAS CIRCULARES SIMPLES
 * El ejercicio consiste en realizar un sorteo, cuya dinámica empieza formando
 * una ronda con los participantes. Para elegir al ganador, dado un número (k),
 * partiendo del inicio se van retirando de la ronda los participantes cuya
 * posición sea igual sea múltiplo de k. Por ejemplo, para k = 3, el primer
 * participante en salir será el de la posición 3, el siguiente será el 6 y así
 * sucesivamente, hasta que quede un solo participante, ese será el ganador.
 */
int main(int argc, char** argv) {

    struct Lista lista;
    construir(lista);
    /*m es la cantidad de jugadores*/
    int m = 12;

    /*A cada jugador se le entrega un ticket con un número, i representa el número
     de ticket*/

    /*INSERTAR AL INICIO DE LA LISTA CIRCULAR*/
//    for(int i = 12; i >= 1; i--)
//        insertarAlInicio(lista, i);
//
    /*INSERTAR AL FINAL DE LA LISTA CIRCULAR*/
    for(int i = 1; i < m + 1; i++)
        insertarAlFinal(lista, i);

    /*INSERTAR EN ORDEN EN LA LISTA CIRCULAR*/
//    insertarEnOrden(lista, 5);
//    insertarEnOrden(lista, 4);
//    insertarEnOrden(lista, 1);
//    insertarEnOrden(lista, 3);
//    insertarEnOrden(lista, 2);

    /*ELIMINAR UN ELEMENTO DADO DE LA LISTA CIRCULAR*/
////    eliminaNodo(lista, 2);
//    eliminaNodo(lista, 4);
    /*Empieza el juego con k = 3, k es el avance para sacar del juego
      un participante*/
    /*RESUELVE EL PROBLEMA DEL SORTEO CIRCULAR- PROBLEMA BASADO EN JOSEFO*/
    problemaJosefo(lista, 3);

    /*Al final solo debe quedar un participante*/
    /*IMPRIME LOS ELEMENTOS DE LA LISTA CIRCULAR*/
    imprime(lista);

    /*LIBERA LA MEMORIA*/
    destruirLista(lista);
    return 0;

}

/*
 * File:   Nodo.h
 * Author: ANA RONCAL
 * Created on 31 de agosto de 2023, 11:15 PM
 */

#ifndef NODO_H
```

```
#define NODO_H

struct Nodo{
    int elemento;
    struct Nodo * siguiente;
};

#endif /* NODO_H */

/*
 * File:   Lista.h
 * Author: ANA RONCAL
 * Created on 31 de agosto de 2023, 11:15 PM
 */

#ifndef LISTA_H
#define LISTA_H
/*LISTAS CIRCULARES SIMPLES*/
/*Listas circulares, la principal ventaja es que permiten la navegación en
     * un sentido a través de la misma, además, se puede recorrer toda la lista
     * partiendo de cualquier elemento (Nodo) siempre que tengamos un apuntador
     * a este. Se deben establecer condiciones adecuadas para evitar caer en
     * ciclos infinitos*/
struct Lista{
    struct Nodo * cabeza; /*apunta al inicio de la lista*/
    int longitud; /*guarda la longitud de la lista*/
};

#endif /* LISTA_H */

/*
 * File:   funcionesLista.h
 * Author: ANA RONCAL
 * Created on 31 de agosto de 2023, 11:16 PM
 */

#ifndef FUNCIONESLISTA_H
#define FUNCIONESLISTA_H

void construir(struct Lista &);
bool esListaVacia(const struct Lista &);
int longitud(const struct Lista & );
struct Nodo * crearNuevoNodo(int , struct Nodo * );
struct Nodo * obtenerUltimoNodo(const struct Lista &);
struct Nodo * obtenerNodoAnterior(const struct Lista &, int );
void insertarAlInicio(struct Lista & , int );
void insertarAlFinal(struct Lista  & , int );
void insertarEnOrden(struct Lista & , int );
void imprime(const struct Lista & );
void problemaJosefo(struct Lista & , int );
void eliminaNodo(struct Lista & , int );
void destruirLista(struct Lista  & );

#endif /* FUNCIONESLISTA_H */

/*
 * File:   funcionesLista.cpp
 * Author: ANA RONCAL
 * Created on 31 de agosto de 2023, 11:17 PM
 */

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstring>
#include "Nodo.h"
#include "Lista.h"
using namespace std;
#include "funcionesLista.h"
```

```
139
140 /*INICIALIZA LOS VALORES DE LA LISTA*/
141 void construir(struct Lista & tad){
142     /*Igual que una lista simplemente enlazada*/
143     tad.cabeza = nullptr;
144     tad.longitud = 0;
145 }
146
147 /*DEVUELVE EL TAMAÑO DE LA LISTA*/
148 int longitud(const struct Lista & tad){
149     return tad.longitud;
150 }
151
152 /*NOS DICE SI SE ENCUENTRAN ELEMENTOS EN LA LISTA*/
153 bool esListaVacia(const struct Lista & tad){
154     /*Igual que una lista simplemente enlazada*/
155     return tad.cabeza == nullptr;
156 }
157
158 /*SE CREA EL ELEMENTO CON VALORES INICIALES*/
159 struct Nodo * crearNodo(int elemento, struct Nodo * siguiente){
160     /*Igual que una lista simplemente enlazada*/
161     struct Nodo * nuevoNodo = new struct Nodo;
162     nuevoNodo->elemento = elemento;
163     nuevoNodo->siguiente = siguiente;
164     return nuevoNodo;
165 }
166
167 /*DEVUELVE EL ÚLTIMO ELEMENTO DE LA LISTA*/
168 struct Nodo * obtenerUltimoNodo(const struct Lista & tad){
169     //si la cabeza es nula entonces no existe ningún elemento en la lista
170     if (esListaVacia(tad)) /*primero preguntamos si hay elementos en la lista*/
171         return nullptr;
172     struct Nodo * ultimo = nullptr;
173     struct Nodo * recorrido = tad.cabeza;
174     do{
175         ultimo = recorrido;
176         recorrido = recorrido->siguiente;
177     } while(recorrido != tad.cabeza); /*CONDICION DIFERENTE A LA LISTA SIMPLE
178                                 cuando llegue a cabeza que acabe el recorrido*/
179     return ultimo;
180 }
181
182 /*INSERTA DESDE CABEZA DE LA LISTA CIRCULAR SIMPLE*/
183 void insertarAlInicio(struct Lista & tad, int elemento){
184     /*se crea el nodo apuntando a la cabeza*/
185     struct Nodo * nuevoNodo = crearNodo(elemento, tad.cabeza);
186     struct Nodo * ultimo;
187
188     if (tad.cabeza == nullptr){
189         /*Como es el primero se apunta a si mismo*/
190         tad.cabeza = nuevoNodo;
191         nuevoNodo->siguiente = nuevoNodo;
192     }
193     else{
194         ultimo = obtenerUltimoNodo(tad);
195         ultimo->siguiente = nuevoNodo;
196         tad.cabeza = nuevoNodo;
197     }
198     tad.longitud++;
199 }
200
201 /*SE INSERTA AL FINAL DE LA LISTA CIRCULAR SIMPLE*/
202 void insertarAlFinal(struct Lista  & tad, int elemento){
203
204     struct Nodo * ultimoNodo = obtenerUltimoNodo(tad);
205     struct Nodo * nuevoNodo = crearNodo(elemento, tad.cabeza);
206     /*se crea el nodo apuntando a la cabeza*/
207
```

```
208         if (ultimoNodo == nullptr){
209             tad.cabeza = nuevoNodo;
210             nuevoNodo->siguiente = nuevoNodo;
211         }
212         else{
213             ultimoNodo->siguiente = nuevoNodo;
214         }
215         tad.longitud++;
216     }
217
218     struct Nodo * obtenerNodoAnterior(const struct Lista & tad, int elemento){
219         if (esListaVacia(tad))
220         return nullptr;
221         struct Nodo * anterior = nullptr;
222         struct Nodo * recorrido = tad.cabeza;
223         do{
224             if(recorrido->elemento < elemento){
225             anterior = recorrido;
226             recorrido = recorrido->siguiente;
227             }                                          /*De menor a mayor*/
228         } while(recorrido != tad.cabeza and recorrido->elemento < elemento);
229         return anterior;
230     }
231
232     void insertarEnOrden(struct Lista & tad, int elemento){
233         struct Nodo * nuevoNodo = crearNodo(elemento, nullptr);
234         struct Nodo * nodoAnterior = obtenerNodoAnterior(tad, elemento);
235
236         if (nodoAnterior == nullptr){
237             struct Nodo * ultimoNodo = obtenerUltimoNodo(tad);
238             nuevoNodo->siguiente = tad.cabeza;
239             tad.cabeza = nuevoNodo;
240             if (ultimoNodo != NULL)
241                 ultimoNodo->siguiente = nuevoNodo;
242             else
243                 nuevoNodo->siguiente = nuevoNodo;
244         }else{
245             nuevoNodo->siguiente = nodoAnterior->siguiente;
246             nodoAnterior->siguiente = nuevoNodo;
247         }
248         tad.longitud++;
249     }
250
251     /*RESULVE EL PROBLEMA DEL SORTEO CIRCULAR*/
252     void problemaJosefo(struct Lista & tad, int i){
253
254         struct Nodo * recorrido = new struct Nodo;
255         struct Nodo * siguiente = new struct Nodo;
256
257         recorrido = tad.cabeza; /*empezamos el recorrido*/
258         int longitud = tad.longitud; /* hallamos la longitud*/
259         do{
260             for(int j = 1; j < i; j++) /*cantidad de veces que avanzo*/
261                 recorrido = recorrido->siguiente;
262             siguiente = recorrido->siguiente; /*guardo el siguiente*/
263             cout<<"Se elimina: "<<recorrido->elemento<<endl;
264             if(recorrido == tad.cabeza) /*guardo la cabeza*/
265                 tad.cabeza = recorrido->siguiente; /*para no perder la cabeza*/
266             eliminaNodo(tad, recorrido->elemento); /*elimino el nodo i en i */
267             recorrido = siguiente;/*paso al siguiente*/
268
269         }while(tad.longitud != 1); /*acabo cuando me quede uno*/
270
271
272     }
273     /*DEVUELVE EL ELEMENTO QUE COINCIDE CON LA BÚSQUEDA*/
274     struct Nodo * seEncuentra(const struct Lista & tad, int llave){
275         if (esListaVacia(tad))
276             return nullptr;
```

```
277     struct Nodo * recorrido = tad.cabeza;
278     do{
279         if (recorrido->elemento == llave) /*SON IGUALES*/
280             return recorrido;
281         recorrido = recorrido->siguiente;
282     } while(recorrido != tad.cabeza);  /*mientras no llegue a cabeza*/
283     return nullptr;         /*En caso que no lo encuentre*/
284 }
285 
286 /*MUESTRA LOS ELEMENTOS DE LA LISTA CIRCULAR SIMPLE*/
287 void imprime(const struct Lista & tad){
288     struct Nodo * recorrido = tad.cabeza;
289     int estaImprimiendoLaCabeza = 1;
290     cout<<"[";
291     if (recorrido != nullptr)
292         do{
293             if(!estaImprimiendoLaCabeza)
294                 cout<<", ";
295             estaImprimiendoLaCabeza = 0;
296             cout<<recorrido->elemento;
297             recorrido = recorrido->siguiente;
298         } while(recorrido != tad.cabeza); /*condición de una vuelta*/
299     cout<<"]\n";
300 }
301 
302 /*ELIMINA UN ELEMENTO DE LA LISTA CIRCULAR, EL ELEMENTO COINCIDE CON LA BÚSQUEDA*/
303 void eliminaNodo(struct Lista & tad, int elemento){
304     struct Nodo * ultimo = nullptr;
305     struct Nodo * recorrido = tad.cabeza;  /*empezamos en cabeza*/
306     int encontrado = 0;
307     do {
308         if (recorrido->elemento == elemento){
309             encontrado = 1;
310             break; /*sale del do si lo encuentra*/
311         }
312         ultimo = recorrido;
313         recorrido = recorrido->siguiente; /*avanza*/
314     } while(recorrido != tad.cabeza); /* busca el elemento */
315 
316     if (encontrado){
317         if (ultimo == nullptr){ /*estoy al inicio de la lista*/
318             if (recorrido == recorrido->siguiente)
319                     tad.cabeza = nullptr; /*elimina la cabeza*/
320             else{
321                 struct Nodo * ultimoNodo = obtenerUltimoNodo(tad);
322                 tad.cabeza = recorrido->siguiente; /*salva la cabeza*/
323                 if (ultimoNodo != nullptr)
324                     ultimoNodo->siguiente = tad.cabeza;
325             }
326         }
327         else{
328             ultimo->siguiente = recorrido->siguiente;
329         }
330         delete recorrido;
331         tad.longitud--;
332     }
333 }
334 
335 /*LIBERA ESPACIO DE MEMORIA*/
336 void destruirLista(struct Lista  & tad){
337     struct Nodo * recorrido = tad.cabeza;
338     if (recorrido == nullptr)
339         return;
340     do{
341         struct Nodo * nodoEliminar = recorrido;
342         recorrido = recorrido->siguiente;
343         delete nodoEliminar;
344     } while(recorrido != tad.cabeza);
345     tad.cabeza = nullptr;
```

```
346         tad.longitud = 0;
347     }
```