Manual Técnico

Brandon Josué Pinto Méndez

201930236

Main:

el main se ocupa de ejecutar el método que inicia el programa:

```
#include "Carta.h"
#include "Manejador.h"


#include <random>    // Para std::default_random_engine y std::uniform_int_distribution
#include <chrono>
int main() {
    Carta* Baraja = new Carta[24];
    Carta* Tablero = new Carta[28];
    Manejador Mj;
    Mj.InsertarCartas(Baraja,Tablero);
    // Liberar memoria de los arreglos
    delete[] Baraja;
    delete[] Tablero;
    return 0;
}
```

Manejador:

El manejador es el principal del programa, aquí están todos los métodos los cuales ejecutan el programa y lo hacen funcionar.

```
#include "Manejador.h"
#include "Carta.h"
#include "NodoCarta.h"
#include <cstdlib> // Para la función rand()
#include <algorithm>
#include <random>
#include <chrono>
#include <iostream>
```

```
Manejador::Manejador() {


};


//en ejecución el primer metodo utilizado, crea los arreglos con las cartas:
void Manejador::InsertarCartas(Carta* Baraja,Carta* Tablero) {
    std::default_random_engine
generator(std::chrono::system_clock::now().time_since_epoch().count());
    int cRepes = 1;
    int cArregloBaraja = 0;
    int cArregloTablero = 0;
    std::string pre = "";
    char color = 'n';
    while (cRepes <= 4) {
        if (cRepes == 1) {
            pre = "E3";
            color = 'n';
        } else if (cRepes == 2) {
            pre = "<>";
            color = 'r';
        } else if (cRepes == 3) {
            pre = "!!";
            color = 'n';
        } else if (cRepes == 4) {
            pre = "<3";
            color = 'r';
        }
        for (int i = 0; i < 13; ++i) {
            double opcion = static_cast<double>(std::rand()) / RAND_MAX;
```

```
            if(opcion <= 0.5){
                if(cArregloBaraja!=24){
                    Baraja[cArregloBaraja] = Carta(i+1, pre, color);
                    cArregloBaraja++;
                }else{
                    Tablero[cArregloTablero] = Carta(i+1, pre, color);
                    cArregloTablero++;
                }
            }else{
                if(cArregloTablero!=28){
                    Tablero[cArregloTablero] = Carta(i+1, pre, color);
                    cArregloTablero++;
                }else{
                    Baraja[cArregloBaraja] = Carta(i+1, pre, color);
                    cArregloBaraja++;
                }
            }
        }
        cRepes++;
    }
    //Revolver las cartas:
    std::shuffle(Baraja,Baraja+24,generator);
    std::shuffle(Tablero,Tablero+28,generator);
    bar = Baraja;
    tab = Tablero;
    PonerCartasTablero();
}
//segundo metodo utilizado:
```

```
void Manejador::PonerCartasTablero(){
    int EspacioMemoria =0;
    EspacioMemoria = asignarCol(&Col1, 1, EspacioMemoria, tab);
    EspacioMemoria = asignarCol(&Col2, 2, EspacioMemoria, tab);
    EspacioMemoria = asignarCol(&Col3, 3, EspacioMemoria, tab);
    EspacioMemoria = asignarCol(&Col4, 4, EspacioMemoria, tab);
    EspacioMemoria = asignarCol(&Col5, 5, EspacioMemoria, tab);
    EspacioMemoria = asignarCol(&Col6, 6, EspacioMemoria, tab);
    asignarCol(&Col7, 7, EspacioMemoria, tab);
    AsignarCola(&Cola1, bar);
    Mov.setCartaMovida(nullptr);
    IniciarJuego();
}
//tercer metodo principal
void Manejador::IniciarJuego(){
    bool Sigue = true;
    std::cout << "------------------------------Bienvenido a Solitario----------------------" <<
std::endl;
    while (Sigue){
        int opcion;
        ImprimirPantalla();
        std::cout << "1) cambiar carta del top" << std::endl;
        std::cout << "2) Mover alguna carta" << std::endl;
        std::cout << "3) Regresar (Muerte)" << std::endl;
        std::cout << "4) Ver Carta" << std::endl;
        std::cout << "5) Finalizar juego" << std::endl;
        std::cout << "Que desea hacer?";
        std::cin >> opcion;
        if(opcion==5){
```

```
            Sigue = false;
        }else{
            Jugar(opcion);
            Sigue = ComprobarVictory();
        }
    }
    if(!ComprobarVictory()){
        std::cout << "Felicidades, has ganado el solitario!!";
    }else{
        std::cout << "Casi lo consigues! suerte a la proxima";
    }
}
//cuarto metodo principal
void Manejador::Jugar(int Opcion){
    Movimiento *aux= new Movimiento();
    int casillaI, casillaF;
    switch (Opcion) {
        case 1:
            if(Cola1.getNodoSig() == nullptr){
                aux->setCartaMovida(Cola1.getCartaApuntada());
                aux->setNodoI(&Cola2);
                aux->setNodoF(&Cola1);
                AgregarNodoDobleMov(&Mov,aux);
                Cola1 = Cola2;
                Cola1.getCartaApuntada()->setOcultar(false);
                Cola2 = *new NodoCarta();
            }else{
                aux->setCartaMovida(Cola1.getCartaApuntada());
```

```
            aux->setNodoI(&Cola1);
            aux->setNodoF(&Cola2);
            AgregarNodoDobleMov(&Mov,aux);
            if(Cola2.getCartaApuntada() == nullptr){
                Cola2.setCartaApuntada(Cola1.getCartaApuntada());
                Cola2.getCartaApuntada()->setOcultar(true);
            }else{
                NodoCarta *Nuevo = new NodoCarta();;
                Nuevo->setCartaApuntada(Cola1.getCartaApuntada());
                AgregarNodo(RecorrerNodo(&Cola2),Nuevo);
            }
            Cola1 = *Cola1.getNodoSig();
        }
        break;
    case 2:
        std::cout << "Escriba el numero de casilla donde se encuentra la carta";
        std::cin >> casillaI;
        std::cout << "Escriba el numero de casilla donde quiere mover la carta";
        std::cin >> casillaF;
        if(casillaI < 13 && casillaI > 0 && casillaF < 13 && casillaF > 0){
            if(casillaI==casillaF){
                std::cout << "No puedes mover una carta al mismo lugar"<<std::endl;
            }else{
                std::cout <<movimientoLegal(casillaI,casillaF)<<std::endl;
            }
        }else{
            std::cout << "escogiste una casilla invalida"<<std::endl;
        }
```

```
            break;
        case 3:
            if(Mov.getNodoSiguiente()!= nullptr){
                Movimiento* aux = RecorrerNodoMov(&Mov);
                NodoCarta *Nuevo= new NodoCarta;
                NodoCarta *ultimoNodo = new NodoCarta;
                if(aux->getNodoI()==&Cola1 && aux->getNodoF()==&Cola2){
                    Nuevo->setCartaApuntada(RecorrerNodo(&Cola2)->getCartaApuntada());
                    Nuevo->getCartaApuntada()->setOcultar(false);
                    Nuevo->setNodoSig(&Cola1);
                    Cola1.setNodoAnterior(Nuevo);
                    Cola1 = *Cola1.getNodoAnterior();
                }else if(aux->getNodoI()==&Cola2 && aux->getNodoF()==&Cola1){

                }else{
                    ultimoNodo= RecorrerNodo(aux->getNodoF());
                    if (ultimoNodo) {
                        Nuevo->setCartaApuntada(ultimoNodo->getCartaApuntada());
                        // Verificar si hay solo un nodo en la lista
                        if (ultimoNodo->getNodoAnterior() == nullptr) {
                            // La lista solo tiene un nodo, eliminarlo
                            aux->getNodoF()->setCartaApuntada(nullptr); // Actualizar el puntero
inicial de la lista
                        } else {
                            // La lista tiene más de un nodo, eliminar el último nodo
                            NodoCarta* nodoAnterior = ultimoNodo->getNodoAnterior();
                            nodoAnterior->setNodoSig(nullptr);
                            nodoAnterior->getCartaApuntada()->setOcultar(false);
                            ultimoNodo->setNodoAnterior(nullptr);
```

```
                }

            }
            RecorrerNodo(aux->getNodoI())->getCartaApuntada()->setOcultar(true);
            AgregarNodoDoble(RecorrerNodo(aux->getNodoI()),Nuevo);
            std::cout << "Movimiento Completado!"<<std::endl;
          }
        }else{
          std::cout << "No tienes movimientos previos"<<std::endl;
        }
        break;
      case 4:
        std::cout << "Escriba el numero de casilla donde se encuentra la carta en el tablero";
        std::cin >> casillaI;
        std::cout << "Escriba el numero de Fila de la carta que quieras ver";
        std::cin >> casillaF;
        if(casillaI>=6&&casillaI<=12&&casillaF>0){
          VerCartas(casillaI,casillaF);
        }else{
          std::cout << "Escoje un valor valido en las opciones"<<std::endl;
        }
        break;
      default:

        break;
    }
}
```

```
bool Manejador::ComprobarVictory(){
   NodoCarta* nodo1 = RecorrerNodo(&Ganar1);
   NodoCarta* nodo2 = RecorrerNodo(&Ganar2);
   NodoCarta* nodo3 = RecorrerNodo(&Ganar3);
   NodoCarta* nodo4 = RecorrerNodo(&Ganar4);


   if (nodo1->getCartaApuntada() != nullptr  && nodo2->getCartaApuntada() != nullptr
&& nodo3->getCartaApuntada() != nullptr && nodo4->getCartaApuntada() != nullptr) {
      if (nodo1->getCartaApuntada()->getValor() == 13 &&
         nodo2->getCartaApuntada()->getValor() == 13 &&
         nodo3->getCartaApuntada()->getValor() == 13 &&
         nodo4->getCartaApuntada()->getValor() == 13) {
         return false;
      }
   }
   return true;
}
//metodo auxiliar del tercer principal para imprimir pantalla
void Manejador::ImprimirPantalla(){
   std::cout << "     1                    2       3       4       5" << std::endl;
   NodoCarta *Mc1 = &Col1, *Mc2 = &Col2, *Mc3 = &Col3, *Mc4 = &Col4, *Mc5 =
&Col5, *Mc6 = &Col6, *Mc7 = &Col7;
   bool Sigue = true;


   std::cout <<Cola1.getCartaApuntada()->getDatos()<<(Cola2.getCartaApuntada() ?
Cola2.getCartaApuntada()->getDatos() : " | | ")<<"                  "
         << (RecorrerNodo(&Ganar1)->getCartaApuntada() ? RecorrerNodo(&Ganar1)-
>getCartaApuntada()->getDatos() : " | | ") << "    "
         << (RecorrerNodo(&Ganar2)->getCartaApuntada() ? RecorrerNodo(&Ganar2)-
>getCartaApuntada()->getDatos() : " | | ") << "    "
```

```
        << (RecorrerNodo(&Ganar3)->getCartaApuntada() ? RecorrerNodo(&Ganar3)-
>getCartaApuntada()->getDatos() : " | | ") << "    "

        << (RecorrerNodo(&Ganar4)->getCartaApuntada() ? RecorrerNodo(&Ganar4)-
>getCartaApuntada()->getDatos() : " | | ") << std::endl;

   std::cout<<std::endl;

   while(Sigue){

      std::cout << (Mc1 && Mc1->getCartaApuntada() ? Mc1->getCartaApuntada()-
>getDatos() : "      ") << "     "

            << (Mc2 && Mc2->getCartaApuntada() ? Mc2->getCartaApuntada()-
>getDatos() : "      ") << "     "

            << (Mc3 && Mc3->getCartaApuntada() ? Mc3->getCartaApuntada()-
>getDatos() : "      ") << "     "

            << (Mc4 && Mc4->getCartaApuntada() ? Mc4->getCartaApuntada()-
>getDatos() : "      ") << "     "

            << (Mc5 && Mc5->getCartaApuntada() ? Mc5->getCartaApuntada()-
>getDatos() : "      ") << "     "

            << (Mc6 && Mc6->getCartaApuntada() ? Mc6->getCartaApuntada()-
>getDatos() : "      ") << "     "

            << (Mc7 && Mc7->getCartaApuntada() ? Mc7->getCartaApuntada()-
>getDatos() : "      ") << std::endl;


      if(Mc1 == nullptr && Mc2 == nullptr && Mc3 == nullptr&&Mc4 == nullptr&&Mc5
== nullptr&&Mc6 == nullptr&& Mc7 == nullptr){

         Sigue = false;

      }else{

         if(Mc1 != nullptr){

            Mc1 = Mc1->getNodoSig();

         }

         if(Mc2 != nullptr){

            Mc2 = Mc2->getNodoSig();

         }

         if(Mc3 != nullptr){
```

```
            Mc3 = Mc3->getNodoSig();
        }
        if(Mc4 != nullptr){
            Mc4 = Mc4->getNodoSig();
        }
        if(Mc5 != nullptr){
            Mc5 = Mc5->getNodoSig();
        }
        if(Mc6 != nullptr){
            Mc6 = Mc6->getNodoSig();
        }
        if(Mc7 != nullptr){
            Mc7 = Mc7->getNodoSig();
        }
    }
  }
  std::cout << "  6         7        8        9        10        11        12" << std::endl;
}
//metodo para ver cartas que haya dicho el usuario
void Manejador::VerCartas(int casilla, int repes){
  NodoCarta *Inicio = obtenerNodoMov(casilla);
  bool llave = false;
  for (int i = 0; i < repes-1; ++i) {
    if(Inicio->getNodoSig()!= nullptr){
        Inicio = Inicio->getNodoSig();
    }else{
        llave = true;
    }
```

```
}
bool VIa = Inicio->getNodoAnterior()->getCartaApuntada()->getOcultar();
bool VDa = Inicio->getNodoSig()->getCartaApuntada()->getOcultar();
if(llave){
   std::cout << "Te pasaste de las cartas en la fila"<<std::endl;
}else{
   std::string c;
   if(Inicio->getNodoSig()== nullptr&&Inicio->getNodoAnterior()== nullptr){
      std::cout << "Solo hay una carta en esta columna!"<<std::endl;
   }else if(Inicio->getNodoSig()== nullptr){

      Inicio->getNodoAnterior()->getCartaApuntada()->setOcultar(false);
      ImprimirPantalla();
      Inicio->getNodoAnterior()->getCartaApuntada()->setOcultar(VIa);
      std::cout << "Mande cualquier valor para continuar  ";
      std::cin >> c;
   }else if(Inicio->getNodoAnterior()== nullptr){
      Inicio->getNodoSig()->getCartaApuntada()->setOcultar(false);
      ImprimirPantalla();
      Inicio->getNodoSig()->getCartaApuntada()->setOcultar(VDa);
      std::cout << "Mande cualquier valor para continuar ";
      std::cin >> c;
   }else{
      Inicio->getNodoAnterior()->getCartaApuntada()->setOcultar(false);
      Inicio->getNodoSig()->getCartaApuntada()->setOcultar(false);
      ImprimirPantalla();
      Inicio->getNodoAnterior()->getCartaApuntada()->setOcultar(VIa);
      Inicio->getNodoAnterior()->getCartaApuntada()->setOcultar(VDa);
```

```
            std::cout << "Mande cualquier valor para continuar ";
            std::cin >> c;


        }
    }
}
//metodo de apoyo del cuarto metodo principal encargado de verificar si se puede asignar una carta al punto puesto
std::string Manejador::movimientoLegal(int inicio, int final){
    NodoCarta *Inicio = obtenerNodo(inicio);
    NodoCarta *Final = obtenerNodo(final);
    NodoCarta* auxI = obtenerNodoMov(inicio);
    auto *aux = new Movimiento();
    auto * CartaN = new Carta();
    switch (final) {
        case 1:
            return "No puedes poner devuelta una carta en la baraja.";
        case 2:
        case 3:
        case 4:
        case 5:
            //condiciones para poder poner la carta en el nuevo lugar
            if(Final->getCartaApuntada()== nullptr){
                if (Inicio->getCartaApuntada()->getValor() == 1) {
                    CartaN = obtenerCartaNodo(inicio);
                    Final->setCartaApuntada(CartaN);
                    aux->setNodoI(auxI);
                    aux->setNodoF(Final);
                    aux->setCartaMovida(CartaN);
```

```
                AgregarNodoDobleMov(&Mov,aux);
                return "Movimiento completado.";
            } else {
                return "Movimiento invalido, tienes que iniciar con un A para meter cartas aqui.";
            }
        }else{
            if(Inicio->getCartaApuntada()->getValor()==(Final->getCartaApuntada()->getValor()+1)&&Inicio->getCartaApuntada()->getPrefijo()==Final->getCartaApuntada()->getPrefijo()){
                CartaN = obtenerCartaNodo(inicio);
                NodoCarta* nuevo = new NodoCarta();
                nuevo->setCartaApuntada(CartaN);
                AgregarNodoDoble(obtenerNodo(final),nuevo);
                aux->setNodoI((auxI));
                aux->setNodoF((Final));
                aux->setCartaMovida(CartaN);
                AgregarNodoDobleMov(&Mov,aux);
                return "Movimiento completado.";
            }else{
                return "Movimiento invalido, haber estudiado";
            }
        }
    case 6:
    case 7:
    case 8:
    case 9:
    case 10:
    case 11:
```

```
        case 12:
            //condiciones para poder poner la carta en el nuevo lugar
            if(Final->getCartaApuntada()== nullptr||(Inicio->getCartaApuntada()-
>getValor()==(Final->getCartaApuntada()->getValor()-1)&&Inicio->getCartaApuntada()-
>getColor()!=Final->getCartaApuntada()->getColor())){
                if(Final->getCartaApuntada()== nullptr){
                    CartaN = obtenerCartaNodo(inicio);
                    Final->setCartaApuntada(CartaN);
                    aux->setNodoI((auxI));
                    aux->setNodoF((Final));
                    aux->setCartaMovida(CartaN);
                    AgregarNodoDobleMov(&Mov,aux);
                    return "Movimiento completado.";
                }else{
                    CartaN = obtenerCartaNodo(inicio);
                    NodoCarta* nuevo = new NodoCarta();
                    nuevo->setCartaApuntada(CartaN);
                    AgregarNodoDoble(obtenerNodo(final),nuevo);
                    aux->setNodoI((auxI));
                    aux->setNodoF((Final));
                    aux->setCartaMovida(CartaN);
                    AgregarNodoDobleMov(&Mov,aux);
                  return "Movimiento completado.";
                }
            }else{
                return "Movimiento invalido, haber estudiado";
            }
        default:
            return "Alguna casilla marcada no existe, revise el tablero bien.";
```

```
    }
}
//metodo de apoyo del metodo de apoyo para obtener el nodo al cual quiere acceder el usuario
Carta * Manejador::obtenerCartaNodo(int numero){
    Carta* enviado = nullptr;
    NodoCarta* ultimoNodo,*NodoActual = new NodoCarta();
    switch (numero) {
        case 1:
            enviado = Cola1.getCartaApuntada();
            if(Cola1.getNodoSig()== nullptr){
                Cola1.setCartaApuntada(nullptr);
            }else{
                Cola1 = *Cola1.getNodoSig();
            }
            return enviado;
        case 2:
            ultimoNodo = RecorrerNodo(&Ganar1);
            NodoActual = &Ganar1;
            break;
        case 3:
            ultimoNodo = RecorrerNodo(&Ganar2);
            NodoActual = &Ganar2;
            break;
        case 4:
            ultimoNodo = RecorrerNodo(&Ganar3);
            NodoActual = &Ganar3;
            break;
        case 5:
```

```
    ultimoNodo = RecorrerNodo(&Ganar4);
    NodoActual = &Ganar4;
    break;
case 6:
    ultimoNodo = RecorrerNodo(&Col1);
    NodoActual = &Col1;
    break;
case 7:
    ultimoNodo = RecorrerNodo(&Col2);
    NodoActual = &Col2;
    break;
case 8:
    ultimoNodo = RecorrerNodo(&Col3);
    NodoActual = &Col3;
    break;
case 9:
    ultimoNodo = RecorrerNodo(&Col4);
    NodoActual = &Col4;
    break;
case 10:
    ultimoNodo = RecorrerNodo(&Col5);
    NodoActual = &Col5;
    break;
case 11:
    ultimoNodo = RecorrerNodo(&Col6);
    NodoActual = &Col6;
    break;
case 12:
```

```
            ultimoNodo = RecorrerNodo(&Col7);
            NodoActual = &Col7;
            break;
        default:
            break;
    }
    if (ultimoNodo) {
        enviado = ultimoNodo->getCartaApuntada();
        // Verificar si hay solo un nodo en la lista
        if (ultimoNodo->getNodoAnterior() == nullptr) {
            // La lista solo tiene un nodo, eliminarlo
            NodoActual->setCartaApuntada(nullptr); // Actualizar el puntero inicial de la lista
        } else {
            // La lista tiene más de un nodo, eliminar el último nodo
            NodoCarta* nodoAnterior = ultimoNodo->getNodoAnterior();
            nodoAnterior->setNodoSig(nullptr);
            nodoAnterior->getCartaApuntada()->setOcultar(false);
            ultimoNodo->setNodoAnterior(nullptr);
            delete ultimoNodo;
        }

    }
    return enviado;
}
//metodo de apoyo del metodo de apoyo para obtener el nodo al cual quiere acceder el usuario
NodoCarta* Manejador::obtenerNodo(int numero){
    switch (numero) {
        case 1:
```

```
            return &Cola1;
        case 2:
            return RecorrerNodo(&Ganar1);
        case 3:
            return RecorrerNodo(&Ganar2);
        case 4:
            return RecorrerNodo(&Ganar3);
        case 5:
            return RecorrerNodo(&Ganar4);
        case 6:
            return RecorrerNodo(&Col1);
        case 7:
            return RecorrerNodo(&Col2);
        case 8:
            return RecorrerNodo(&Col3);
        case 9:
            return RecorrerNodo(&Col4);
        case 10:
            return RecorrerNodo(&Col5);
        case 11:
            return RecorrerNodo(&Col6);
        case 12:
            return RecorrerNodo(&Col7);
        default:
            break;
    }
}
//metodo de apoyo del metodo de apoyo para obtener el nodo al cual quiere acceder el
usuario
```

```
NodoCarta* Manejador::obtenerNodoMov(int numero){
    switch (numero) {
        case 1:
            return &Cola1;
        case 2:
            return &Ganar1;
        case 3:
            return &Ganar2;
        case 4:
            return &Ganar3;
        case 5:
            return &Ganar4;
        case 6:
            return (&Col1);
        case 7:
            return (&Col2);
        case 8:
            return (&Col3);
        case 9:
            return (&Col4);
        case 10:
            return (&Col5);
        case 11:
            return (&Col6);
        case 12:
            return (&Col7);
        default:
            break;
```

```
    }
}

//metodo de apoyo del metodo de apoyo para obtener el nodo al cual quiere acceder el usuario
NodoCarta Manejador::GuardadorDeCambio(int numero){
    switch (numero) {
        case 1:
            return Cola1;
        case 2:
            return RecorrerNodoU(Ganar1);
        case 3:
            return RecorrerNodoU(Ganar2);
        case 4:
            return RecorrerNodoU(Ganar3);
        case 5:
            return RecorrerNodoU(Ganar4);
        case 6:
            return RecorrerNodoU(Col1);
        case 7:
            return RecorrerNodoU(Col2);
        case 8:
            return RecorrerNodoU(Col3);
        case 9:
            return RecorrerNodoU(Col4);
        case 10:
            return RecorrerNodoU(Col5);
        case 11:
            return RecorrerNodoU(Col6);
```

```
        case 12:
            return RecorrerNodoU(Col7);
        default:
            break;
    }
}
//metodo de apoyo del segundo metodo principal encargado de Asignar cola
void Manejador::AsignarCola(NodoCarta* Centinela, Carta* tab){
    for (int i = 0; i < 24; ++i) {
        if(i == 0){
            tab[i].setOcultar(false);
            Centinela->setCartaApuntada(&tab[i]);
        }else{
            NodoCarta* nuevo = new NodoCarta();
            tab[i].setOcultar(false);
            nuevo->setCartaApuntada(&tab[i]);
            NodoCarta* Final = RecorrerNodo(Centinela);
            AgregarNodo(Final,nuevo);
        }
    }
}

//metodo de apoyo del segundo metodo principal
int Manejador::asignarCol(NodoCarta* Centinela, int repes, int contador, Carta* tab){
    for (int i = 0; i < repes; ++i) {
        if(repes == 1){
            tab[contador].setOcultar(false);
            Centinela->setCartaApuntada(&tab[contador]);
```

```
        }else{
            if(i==0){
                Centinela->setCartaApuntada(&tab[contador]);
            }else{
                if(i==repes-1){tab[contador].setOcultar(false);}
                NodoCarta* nuevo = new NodoCarta();;
                nuevo->setCartaApuntada(&tab[contador]);
                NodoCarta* Final = RecorrerNodo(Centinela);
                AgregarNodoDoble(Final,nuevo);
            }
        }
        contador++;
    }
    return contador;
}

//metodos de apoyo para el manejo de los nodos
void Manejador::AgregarNodoDoble(NodoCarta* temp, NodoCarta* Nuevo){
    temp->setNodoSig(Nuevo);
    Nuevo->setNodoAnterior(temp);
}
void Manejador::AgregarNodoDobleMov(Movimiento* temp, Movimiento* Nuevo){
    temp->setNodoSiguiente(Nuevo);
    Nuevo->setNodoAnterior(temp);
}
void Manejador::AgregarNodo(NodoCarta* temp, NodoCarta* Nuevo){
    temp->setNodoSig(Nuevo);
}
```

```
void Manejador::EliminarNodoDoble(NodoCarta* Eliminado){
    if(Eliminado->getNodoSig()== nullptr){
        Eliminado->getNodoAnterior()->setNodoSig(nullptr);
        Eliminado->setNodoAnterior(nullptr);
        delete Eliminado;
    }
}
NodoCarta* Manejador::RecorrerNodo(NodoCarta* Centinela){

    while (Centinela->getNodoSig() != nullptr) {
        Centinela = Centinela->getNodoSig();
    }

    return Centinela;
}
Movimiento* Manejador::RecorrerNodoMov(Movimiento* Centinela){

    while (Centinela->getNodoSiguiente() != nullptr) {
        Centinela = Centinela->getNodoSiguiente();
    }

    return Centinela;
}
NodoCarta Manejador::RecorrerNodoU(NodoCarta Centinela){

    while (Centinela.getNodoSig() != nullptr) {
        Centinela = *Centinela.getNodoSig();
    }
```

```
    return Centinela;
}

void Manejador::RecorrerNodoImprimiendo(NodoCarta* Centinela){
    int i = 1;
    while (Centinela->getNodoSig() != nullptr) {
        Carta *a = Centinela->getCartaApuntada();
        Centinela = Centinela->getNodoSig();
        std::cout << "Cartas en fila[" << i << "]: " << a->getValor()<< " | " << a->getPrefijo()
<< std::endl;
        i++;
    }
    Carta *a = Centinela->getCartaApuntada();
    std::cout << "Cartas en fila[" << i << "]: " << a->getValor()<< " | " << a->getPrefijo() <<
std::endl;
}
//metodo para limpiar pantalla
void Manejador::limpiarPantalla() {
#ifdef _WIN32
    system("cls"); // Para sistemas Windows
#else
    system("clear"); // Para sistemas Unix (Linux/MacOS)
#endif
}
```

Carta:

Esta es una clase objeto y nos ayuda a representar las cartas que utiliza el manejador para el juego.

```
#include <sstream>
```

```
#include "Carta.h"
Carta::Carta(int valor, std::string pre, char color) {
    Prefijo = pre;
    Valor = valor;
    Color = color;
    Ocultar = true;
}
// Método "getter" para el atributo Prefijo
std::string Carta::getPrefijo() const {
    return Prefijo;
}

// Método "getter" para el atributo Valor
int Carta::getValor() const {
    return Valor;
}

// Método "getter" para el atributo Color
char Carta::getColor() const {
    return Color;
}
// Método "getter" para el atributo Ocultar
bool Carta::getOcultar() const {
    return true;
}
//metodo get para buscar todos los datos para imprimir en consola
std::string Carta::getDatos() const{
    std::ostringstream ss;
```

```
    if(Ocultar){
        ss <<"  []  ";
    }else{
        if(Valor==11){
            ss <<" "<<"J"<<Prefijo << Color<<" ";
        }else if(Valor==12){
            ss <<" "<<"Q"<<Prefijo << Color<<" ";
        }else if(Valor==13){
            ss <<" "<<"K"<<Prefijo << Color<<" ";
        }else if(Valor==1){
            ss <<" "<<"A"<<Prefijo << Color<<" ";
        }else if(Valor==10){
            ss <<Valor<<Prefijo << Color<<" ";
        }else{
            ss<<" "<< Valor <<Prefijo << Color<<" ";
        }

    }
    return ss.str();
}
//metodos set
void Carta::setOcultar(bool siguiente) {
    Ocultar = siguiente;
}
```

Nodo Carta: Esta clase se encarga de manejar los nodos de las cartas para poder crear las distintas estructuras que se usan para el manejo del programa, esta clase cuenta con dos punteros a otros nodo carta y un puntero a una carta.

```
#include "NodoCarta.h"
// Constructor
```

```
NodoCarta::NodoCarta(){
    NodoAnterior=nullptr;
    CartaApuntada=nullptr;
    NodoSig=nullptr;
}


// Métodos get

NodoCarta* NodoCarta::getNodoAnterior() const {
   return NodoAnterior;
}

Carta* NodoCarta::getCartaApuntada() const {
   return (CartaApuntada == nullptr) ? nullptr : CartaApuntada;
}

NodoCarta* NodoCarta::getNodoSig() const {
   return NodoSig;
}

// Métodos set

void NodoCarta::setNodoAnterior(NodoCarta* anterior) {
   NodoAnterior = anterior;
}

void NodoCarta::setCartaApuntada(Carta* carta) {
```

```
    CartaApuntada = carta;
}

void NodoCarta::setNodoSig(NodoCarta* siguiente) {
    NodoSig = siguiente;
}
```