```
#include <iostream>
#include <string.h>

#include "ED/lista.h"
#include "loadMembresia.h"
#include "cargarArchivosEnLista.h"
#include "Usuario/usuario.h"
#include "Vinos/Vinos.h"

#include "marketing.h"

/*-----------------------------------------------------------------------------------------------------------
                                    Ranking general de vinos del ultimo año
 -----------------------------------------------------------------------------------------------------------
*/
/*
   PRE: Deben existir los 2 elementos que deseo comparar.
   POST: Comparo los datos e indico si son mayores, menores e iguales.
 */
int comparadorContadorRanking(ELEMENTO elemento1, ELEMENTO elemento2) {
   int iResultado;
   if (((DatoRanking*) elemento1)->contador < ((DatoRanking*) elemento2)->contador)
{
      iResultado = MENOR;
   } else if (((DatoRanking*) elemento1)->contador > ((DatoRanking*) elemento2)-
>contador) {
      iResultado = MAYOR;
   } else
      iResultado = IGUAL;

   return iResultado;
```

```
}

/*
   PRE: Debe existir la lista.
   POST: Busca en la lista dos elementos en los cuales coincida el id_vino.
 */
DatoRanking* findInList(Lista *lista, std::string id_vino) {
   DatoRanking *encontrado = 0;

   for(int i = 0; i < getCantidadDeElementosEnLaLista(lista) && !(bool)encontrado;
i++) {
      ELEMENTO voidElement;
      obtenerElementoDeLaLista(lista, i, voidElement);

      if(stoi(((DatoRanking*)voidElement)->id_vino) == stoi(id_vino))
         encontrado = (DatoRanking*)voidElement;
   }

   return encontrado;
}

/*
   PRE: dato debe ser un puntero al id de un vino, elemento debe ser el contenido de un
nodo de una lista
   POST: Devuelvo true si ambos datos son iguales.
 */
bool compararIdVino(ELEMENTO dato, ELEMENTO elemento) {
   return stoi(*(std::string*)dato) == stoi(((eVinos*)elemento)->id);
}
```

```
Lista* listaParaHacerLosRankings(Lista *listaAnioMembresias, Lista *listaVinos, int &maxYear) {
    maxYear = 0;
    Lista *listaRanking = crearLista();
    Lista *listaMembresias = NULL;

    //Se busca el ultimo año
    for(int i = 0; i < getCantidadDeElementosEnLaLista(listaAnioMembresias); i++) {
        ELEMENTO innerList;
        obtenerElementoDeLaLista(listaAnioMembresias, i, innerList);

        if(stoi(getYearOfList(innerList)) > maxYear) {
            maxYear = stoi(getYearOfList(innerList));
            listaMembresias = (Lista*)innerList;
        }
    }

    //Se recorre la lista anidada del ultimo año
    for(int i = 0; i < getCantidadDeElementosEnLaLista(listaMembresias); i++) {
        ELEMENTO membresia, vinito;
        obtenerElementoDeLaLista(listaMembresias, i, membresia);

        std::string idVinoArr[] {
            ((Membresia*)membresia)->id_vino_1,
            ((Membresia*)membresia)->id_vino_2,
            ((Membresia*)membresia)->id_vino_3,
            ((Membresia*)membresia)->id_vino_4,
            ((Membresia*)membresia)->id_vino_5,
            ((Membresia*)membresia)->id_vino_6
        };
```

```
    //Se recorre cada uno de los 6 vinos de la membresia
    for(int i = 0; i < 6; i++) {
      int posicionVino = buscarElementoEnLaLista(listaVinos, &idVinoArr[i],
compararIdVino);

      //Si existe el vino en el catalogo, se obtienen todos los datos
      if (posicionVino != -1) {
        obtenerElementoDeLaLista(listaVinos, posicionVino, vinito);

        //Se busca si el vino ya se encuentra en el ranking, si no existe, se lo crea y
agrega al ranking
        if(!findInList(listaRanking, idVinoArr[i])) {
          DatoRanking *vino = new DatoRanking();
          vino->id_vino = idVinoArr[i];
          vino->contador++;
          vino->etiqueta_vino = getEtiqueta((eVinos*)vinito);
          vino->nombre_bodega = getBodega((eVinos*)vinito);

          insertarElementoAlFinalDeLaLista(listaRanking, vino);

        }
        //Si existe, solo se incrementa su contador
        else {
          DatoRanking *vinoEnRanking = findInList(listaRanking, idVinoArr[i]);
          vinoEnRanking->contador++;
        }
      }
    }
  }

  return listaRanking;
```

```
}

void rankingVinosUltimoAnio(Lista* listaRanking,  int maxYear){
   int contadorTotalVinos = 0;

   std::cout << "Ranking de vinos (" << maxYear << ")" << std::endl;
   std::cout << "Puesto\t" << "ID\t"<< "Etiqueta\t" << "  Cantidad\t" << std::endl;
   //Ordena la lista del ranking
   reordenarLista(listaRanking, comparadorContadorRanking, descendente);
   for(int i = 0; i < getCantidadDeElementosEnLaLista(listaRanking); i++) {
      ELEMENTO vino;
      obtenerElementoDeLaLista(listaRanking, i, vino);

      std::cout << i+1 << '\t' << ((DatoRanking*)vino)->id_vino << '\t' <<
((DatoRanking*)vino)->etiqueta_vino << "  \t\t" << ((DatoRanking*)vino)->contador
<<std::endl;

      contadorTotalVinos += ((DatoRanking*)vino)->contador;
   }

   std::cout << "\nTotal vinos: " << contadorTotalVinos << std::endl;
}

/*-------------------------------------------------------------------------------------------------------

                                Ranking por bodegas del ultimo año

 -------------------------------------------------------------------------------------------------------
*/
DatoRanking* findInListByBodega(Lista *lista, std::string bodega) {
   DatoRanking *encontrado = 0;
   for(int i = 0; i < getCantidadDeElementosEnLaLista(lista) && !(bool)encontrado;
i++) {
      ELEMENTO voidElement;
```

```
        obtenerElementoDeLaLista(lista, i, voidElement);

        if((((DatoRanking*)voidElement)->nombre_bodega).compare(bodega) == 0)
            encontrado = (DatoRanking*)voidElement;
    }

    return encontrado;
}

void rankingBodegasUltimoAnio(Lista *listaRankingVinos, int maxYear) {
    int contadorVinos = 0;
    Lista* rankingBodegas = crearLista();

    for(int i = 0; i < getCantidadDeElementosEnLaLista(listaRankingVinos); i++) {
        ELEMENTO vino;
        obtenerElementoDeLaLista(listaRankingVinos, i, vino);

        if(!findInListByBodega(rankingBodegas, ((DatoRanking*)vino)-
>nombre_bodega)){
            DatoRanking *bodega = new DatoRanking();
            bodega->nombre_bodega = ((DatoRanking*)vino)->nombre_bodega;
            bodega->contador+= ((DatoRanking*)vino)->contador;

            insertarElementoAlFinalDeLaLista(rankingBodegas, bodega);

        }
        else {
            DatoRanking *bodegaEnRanking = findInListByBodega(rankingBodegas,
((DatoRanking*)vino)->nombre_bodega);
            bodegaEnRanking->contador+= ((DatoRanking*)vino)->contador;
        }
```

```
    }

    std::cout << "\nRanking bodegas (" << maxYear << ")" << std::endl;
    std::cout << "Posición\t" << "Bodega\t\t\t" << "Cantidad de vinos"<< std::endl;
    reordenarLista(rankingBodegas, comparadorContadorRanking, descendente);
    for(int i = 0; i < getCantidadDeElementosEnLaLista(rankingBodegas); i++) {
        ELEMENTO bodegaR;
        obtenerElementoDeLaLista(rankingBodegas, i, bodegaR);

        std::cout << i+1 << "\t\t" << ((DatoRanking*)bodegaR)->nombre_bodega << "\t\t"
<< ((DatoRanking*)bodegaR)->contador << std::endl;
        contadorVinos += ((DatoRanking*)bodegaR)->contador;
    }

    std::cout << "\nTotal vinos: " << contadorVinos << std::endl;
};

/*-------------------------------------------------------------------------------------------------------
                                Ranking de varietales elegido por rango etario
 -------------------------------------------------------------------------------------------------------
*/
//-----------------------------------------------DatoCliente-----------------------------------------
struct DatoCliente{
    Usuario* cliente;
    int iCantidad_De_Vinos_Comprados_Del_Varietal;
};

//-----------------------------------------------Constructor-----------------------------------------
/*
    PRE: El DatoCliente no debe haber sido creado.
    POST: El DatoCliente queda creado.
```

```
 */
DatoCliente* crearDatoCliente(Usuario* cliente){
   DatoCliente* d = new DatoCliente;

   d->cliente = cliente;
   d->iCantidad_De_Vinos_Comprados_Del_Varietal = 1;

   return d;
}

//----------------------------------------------Destructor----------------------------------------------
/*
   PRE: El DatoCliente debe haber sido creado.
   POST: El DatoClienteEsEliminado.
 */
void destruirDatoCliente(DatoCliente* d){
   delete d;
}

//---------------------------------------DatoVarietalPorGrupoEtario---------------------------------
struct DatoVarietalPorGrupoEtario {
   std::string sNombre_Del_Varietal;
   Lista* menoresDe30;
   Lista* entre30Y50;
   Lista* mayoresDe50;
   int iCantidad_De_Ventas_Menores_De_30;
   int iCantidad_De_Ventas_Entre_30_Y_50;
   int iCantidad_De_Ventas_Mayores_De_50;
};
```

```
//-------------------------------------------------Constructor-----------------------------------------


/*
   PRE: El datoVarietalPorGrupoEtario no debe haber sido creado.
   POST: El datoVarietalPorGrupoEtario queda creado.
 */
DatoVarietalPorGrupoEtario* crearDatoDeVarietal(std::string sNombre_Del_Varietal)
{
   DatoVarietalPorGrupoEtario* d = new DatoVarietalPorGrupoEtario();


   d->sNombre_Del_Varietal = sNombre_Del_Varietal;
   d->menoresDe30 = crearLista();
   d->entre30Y50 = crearLista();
   d->mayoresDe50 = crearLista();
   d->iCantidad_De_Ventas_Menores_De_30 = 0;
   d->iCantidad_De_Ventas_Entre_30_Y_50 = 0;
   d->iCantidad_De_Ventas_Mayores_De_50 = 0;


   return d;
}


//-------------------------------------------------Destructor------------------------------------------
/*
   PRE: Debe existir el ELEMENTO que deseo eliminar el cual debe ser un
DatoCliente.
   POST: Elimino el dato del cliente.
 */
void eliminarDatoCliente(ELEMENTO temp) {
   destruirDatoCliente((DatoCliente*) temp);
}
```

```
/*
   PRE: El datoVarietalPorGrupoEtario debe haber sido creado.
   POST: El datoVarietalPorGrupoEtario es eliminado.
 */
void destruirDatoDeVarietal(DatoVarietalPorGrupoEtario* d) {
   destruirListaYDatos(d->menoresDe30, eliminarDatoCliente);
   destruirListaYDatos(d->entre30Y50, eliminarDatoCliente);
   destruirListaYDatos(d->mayoresDe50, eliminarDatoCliente);
   delete d;
}

//-----------------------------------------Funciones extras-----------------------------------------


//---------------------------------------Encontrar elementos---------------------------------------
/*
   PRE: Paso dos datos a la funcion del tipo vino, el primero es el que deseo buscar.
   POST: Devuelvo true si ambos datos son iguales.
 */
bool compararVarietalDelVino(ELEMENTO dato, ELEMENTO elemento) {
   return (getVarietal((eVinos*) dato) == getVarietal((eVinos*) elemento)) ? true : false;
}

/*
   PRE: El primer ELEMENTO debe ser un Usuario y el segundo debe ser un DatoCliente, el primero es el que deseo buscar.
   POST: Devuelvo true si ambos datos son iguales.
 */
bool compararUsuario(ELEMENTO dato, ELEMENTO elemento) {
   return (((Usuario*) dato) == ((DatoCliente*) elemento)->cliente) ? true : false;
}
```

```
/*
   PRE: El primer ELEMENTO debe ser un const char y el segundo debe ser un usuario/vino (segun corresponda), el primero es el que deseo buscar.
   POST: Devuelvo true si ambos datos son iguales.
 */
bool compararIDDelUsuarioOVinoConElDeLaMembresia(ELEMENTO dato, ELEMENTO elemento) {
   return (getID((Usuario*) elemento).compare((const char*) dato) == IGUAL) ? true : false;
}

/*
   PRE: El primer ELEMENTO debe ser un vino y el segundo debe ser un DatoVarietalPorGrupoEtario, el primero es el que deseo buscar.
   POST: Devuelvo true si ambos datos son iguales.
 */
bool compararNombreDelVarietalDelVinoConElDelDatoGrupoEtario(ELEMENTO dato, ELEMENTO elemento) {
   return (getVarietal((eVinos*) dato) == ((DatoVarietalPorGrupoEtario*) elemento)->sNombre_Del_Varietal) ? true : false;
}

/*
   PRE: Debe existir la lista de catalogos.
   POST: Devuelvo la lista de los varietales que existen en el catalogo.
 */
Lista* varietalesQueHay(Lista* lCatalogo) {
   Lista* lAux = crearLista();
   Lista* lVarietales = crearLista();
   ELEMENTO vino;
```

```
    for (int i = 0; i < getCantidadDeElementosEnLaLista(lCatalogo); i++) {
        obtenerElementoDeLaLista(lCatalogo, i, vino);

        if (buscarElementoEnLaLista(lAux, (eVinos*) vino, compararVarietalDelVino) ==
ELEMENTO_NO_ENCONTRADO)
            insertarElementoAlFinalDeLaLista(lAux, (eVinos*) vino);

    }

    ELEMENTO temp;
    for (int i = 0; i < getCantidadDeElementosEnLaLista(lAux); i++) {
        obtenerElementoDeLaLista(lAux, i, vino);
        temp = crearDatoDeVarietal(getVarietal((eVinos*) vino));
        insertarElementoAlFinalDeLaLista(lVarietales, (DatoVarietalPorGrupoEtario*)
temp);
    }

    destruirLista(lAux);

    return lVarietales;
}
/*
    PRE: Debe existir el DatoCliente y la lista del grupo etario.
    POST: Comparo si el DatoCliente existe o no en la lista y si no esta en la lista lo
incorporo.
 */
void insertarUsuarioEnLaListaDeSuGrupoSiNoExisteYaEnElla(Lista* lGrupo_Etario,
DatoCliente* cliente) {
    ELEMENTO dato;

    if (buscarElementoEnLaLista(lGrupo_Etario, cliente->cliente, compararUsuario) !=
ELEMENTO_NO_ENCONTRADO) {
```

```
        obtenerElementoDeLaLista(lGrupo_Etario,
buscarElementoEnLaLista(lGrupo_Etario, cliente->cliente, compararUsuario), dato);

        destruirDatoCliente(cliente);

        ((DatoCliente*) dato)->iCantidad_De_Vinos_Comprados_Del_Varietal++;


    } else

        insertarElementoAlFinalDeLaLista(lGrupo_Etario, (DatoCliente*) cliente);


}


/*

    PRE: Indico el grupo etario del cliente, el varietal y el usuario con el que se debe
operar.

    POST: Opero la lista del grupo etario que corresponda agregando el usuario que
compro un X varietal si es que el usuario no ha comprado previamente dicho varietal e
incremento la cantidad de ventas del varietal en su grupo en 1.

 */

void insertarUsuarioEnLaListaDeSuGrupo(std::string sGrupo_Etario,
DatoVarietalPorGrupoEtario* varietales, Usuario* usuario) {


    DatoCliente* cliente = crearDatoCliente(usuario);


    if (sGrupo_Etario == "menoresDe30") {


        varietales->iCantidad_De_Ventas_Menores_De_30++;

        insertarUsuarioEnLaListaDeSuGrupoSiNoExisteYaEnElla(varietales-
>menoresDe30, cliente);


    } else if (sGrupo_Etario == "entre30Y50") {


        varietales->iCantidad_De_Ventas_Entre_30_Y_50++;
```

```
        insertarUsuarioEnLaListaDeSuGrupoSiNoExisteYaEnElla(varietales->entre30Y50, cliente);


    } else {


        varietales->iCantidad_De_Ventas_Mayores_De_50++;

        insertarUsuarioEnLaListaDeSuGrupoSiNoExisteYaEnElla(varietales->mayoresDe50, cliente);


    }


}


/*

    PRE: Debe existir la lista de varietal, de usuarios, de calatogo, debe existir la membresia y debo indicar el vino que deseo obtener de los que existen.

    POST: Inserto cada vino comprado en el grupo etario que corresponda en base al comprador.

 */

void identificarVarietalDelVino(Membresia* membresia, int iNumero_De_Vino, Lista* lUsuario, Lista* lCatalogos, Lista* lVarietales) {

    ELEMENTO vino, usuario, varietal;


    obtenerElementoDeLaLista(lCatalogos, buscarElementoEnLaLista(lCatalogos, (const char**)getIDVinoDeLaMembresia(membresia, iNumero_De_Vino).c_str(), compararIDDelUsuarioOVinoConElDeLaMembresia), vino);

    obtenerElementoDeLaLista(lUsuario, buscarElementoEnLaLista(lUsuario, (const char**)getIDDelUsuarioDeLaMembresia(membresia).c_str(), compararIDDelUsuarioOVinoConElDeLaMembresia), usuario);

    obtenerElementoDeLaLista(lVarietales, buscarElementoEnLaLista(lVarietales, vino, compararNombreDelVarietalDelVinoConElDelDatoGrupoEtario), varietal);


    if (getEdadUsuario((Usuario*) usuario) < 30)
```

```
        insertarUsuarioEnLaListaDeSuGrupo("menoresDe30",
(DatoVarietalPorGrupoEtario*) varietal, (Usuario*) usuario);


    if (getEdadUsuario((Usuario*) usuario) >= 30 && getEdadUsuario((Usuario*)
usuario) <= 50)

        insertarUsuarioEnLaListaDeSuGrupo("entre30Y50",
(DatoVarietalPorGrupoEtario*) varietal, (Usuario*) usuario);


    if (getEdadUsuario((Usuario*) usuario) > 50)

        insertarUsuarioEnLaListaDeSuGrupo("mayoresDe50",
(DatoVarietalPorGrupoEtario*) varietal, (Usuario*) usuario);

}


//-----------------------------------Funciones de ordenar ranking---------------------------------


/*

    PRE: Deben existir los 2 elementos que deseo comparar.

    POST: Comparo los datos e indico si son mayores, menores e iguales.

 */

int compararMenoresDe30Anios(ELEMENTO elemento1, ELEMENTO elemento2) {

    int iResultado;

    if (((DatoVarietalPorGrupoEtario*) elemento1)-
>iCantidad_De_Ventas_Menores_De_30 < ((DatoVarietalPorGrupoEtario*)
elemento2)->iCantidad_De_Ventas_Menores_De_30) {

        iResultado = MENOR;

    } else if (((DatoVarietalPorGrupoEtario*) elemento1)-
>iCantidad_De_Ventas_Menores_De_30 > ((DatoVarietalPorGrupoEtario*)
elemento2)->iCantidad_De_Ventas_Menores_De_30) {

        iResultado = MAYOR;

    } else

        iResultado = IGUAL;


    return iResultado;
```

```
}

/*
   PRE: Deben existir los 2 elementos que deseo comparar.
   POST: Comparo los datos e indico si son mayores, menores e iguales.
 */
int compararEntre30Y50Anios(ELEMENTO elemento1, ELEMENTO elemento2) {
   int iResultado;
   if (((DatoVarietalPorGrupoEtario*) elemento1)->iCantidad_De_Ventas_Entre_30_Y_50 < ((DatoVarietalPorGrupoEtario*) elemento2)->iCantidad_De_Ventas_Entre_30_Y_50) {
      iResultado = MENOR;
   } else if (((DatoVarietalPorGrupoEtario*) elemento1)->iCantidad_De_Ventas_Entre_30_Y_50 > ((DatoVarietalPorGrupoEtario*) elemento2)->iCantidad_De_Ventas_Entre_30_Y_50) {
      iResultado = MAYOR;
   } else
      iResultado = IGUAL;

   return iResultado;
}

/*
   PRE: Deben existir los 2 elementos que deseo comparar.
   POST: Comparo los datos e indico si son mayores, menores e iguales.
 */
int compararMayoresDe50Anios(ELEMENTO elemento1, ELEMENTO elemento2) {
   int iResultado;
   if (((DatoVarietalPorGrupoEtario*) elemento1)->iCantidad_De_Ventas_Mayores_De_50 < ((DatoVarietalPorGrupoEtario*) elemento2)->iCantidad_De_Ventas_Mayores_De_50) {
      iResultado = MENOR;
```

```
   } else if (((DatoVarietalPorGrupoEtario*) elemento1)-
>iCantidad_De_Ventas_Mayores_De_50 > ((DatoVarietalPorGrupoEtario*)
elemento2)->iCantidad_De_Ventas_Mayores_De_50) {

      iResultado = MAYOR;

   } else

      iResultado = IGUAL;


   return iResultado;

}


/*

   PRE: Deben existir los 2 elementos que deseo comparar.

   POST: Comparo los datos e indico si son mayores, menores e iguales.

 */

int comparar ComprasClientes(ELEMENTO elemento1, ELEMENTO elemento2) {

   int iResultado;

   if (((DatoCliente*)elemento1)->iCantidad_De_Vinos_Comprados_Del_Varietal <
((DatoCliente*)elemento2)->iCantidad_De_Vinos_Comprados_Del_Varietal) {

      iResultado = MENOR;

   } else if (((DatoCliente*)elemento1)->iCantidad_De_Vinos_Comprados_Del_Varietal
> ((DatoCliente*)elemento2)->iCantidad_De_Vinos_Comprados_Del_Varietal) {

      iResultado = MAYOR;

   } else

      iResultado = IGUAL;


   return iResultado;

}


//-------------------------------------------------Mostrar datos----------------------------------------

/*

   PRE: Debe existir la lista de clientes que compraron el varietal del
DatoVarietalPorGrupoEtario.
```

```
   POST: Imprimo los datos de los clientes que compraron dicho varietal.
 */
void mostrarClientesEnElGrupoEtario(Lista* lista) {
   ELEMENTO cliente;

   reordenarLista(lista, compararComprasClientes, descendente);
   for (int i = 0; i < getCantidadDeElementosEnLaLista(lista); i++) {
      obtenerElementoDeLaLista(lista, i, cliente);
      std::cout << "\t\t" << ((DatoCliente*) cliente)->cliente->sID << ": " <<
((DatoCliente*) cliente)->cliente->enNombre->sApellido << ", " << ((DatoCliente*)
cliente)->cliente->enNombre->sNombre << " tiene " << ((DatoCliente*) cliente)-
>cliente->iEdad << " años y compró " << ((DatoCliente*) cliente)-
>iCantidad_De_Vinos_Comprados_Del_Varietal << " vinos de este varietal." <<
std::endl;
   }

}

/*
   PRE: Debe existir el elemento que deseo mostrar y debe ser un
DatoVarietalPorGrupoEtario.
   POST: Imprimo los datos del varietal en dicho grupo etario.
 */
void menoresDe30Anios(ELEMENTO varietal) {
   std::cout << ((DatoVarietalPorGrupoEtario*) varietal)->sNombre_Del_Varietal <<
": " << std::endl;
   std::cout << "\tCantidad de ventas: " << ((DatoVarietalPorGrupoEtario*) varietal)-
>iCantidad_De_Ventas_Menores_De_30 << std::endl;
   std::cout << "\tCantidad de personas: " <<
getCantidadDeElementosEnLaLista(((DatoVarietalPorGrupoEtario*) varietal)-
>menoresDe30) << std::endl;
   mostrarClientesEnElGrupoEtario(((DatoVarietalPorGrupoEtario*) varietal)-
>menoresDe30);
   std::cout << LINEA << std::endl;
```

```
}

/*
   PRE: Debe existir el elemento que deseo mostrar y debe ser un DatoVarietalPorGrupoEtario.
   POST: Imprimo los datos del varietal en dicho grupo etario.
 */
void entre30Y50Anios(ELEMENTO varietal) {
   std::cout << ((DatoVarietalPorGrupoEtario*) varietal)->sNombre_Del_Varietal << ": " << std::endl;
   std::cout << "\tCantidad de ventas: " << ((DatoVarietalPorGrupoEtario*) varietal)->iCantidad_De_Ventas_Entre_30_Y_50 << std::endl;
   std::cout << "\tCantidad de personas: " << getCantidadDeElementosEnLaLista(((DatoVarietalPorGrupoEtario*) varietal)->entre30Y50) << std::endl;
   mostrarClientesEnElGrupoEtario(((DatoVarietalPorGrupoEtario*) varietal)->entre30Y50);
   std::cout << LINEA << std::endl;
}

/*
   PRE: Debe existir el elemento que deseo mostrar y debe ser un DatoVarietalPorGrupoEtario.
   POST: Imprimo los datos del varietal en dicho grupo etario.
 */
void mayoresDe50Anios(ELEMENTO varietal) {
   std::cout << ((DatoVarietalPorGrupoEtario*) varietal)->sNombre_Del_Varietal << ": " << std::endl;
   std::cout << "\tCantidad de ventas: " << ((DatoVarietalPorGrupoEtario*) varietal)->iCantidad_De_Ventas_Mayores_De_50 << std::endl;
   std::cout << "\tCantidad de personas: " << getCantidadDeElementosEnLaLista(((DatoVarietalPorGrupoEtario*) varietal)->mayoresDe50) << std::endl;
   mostrarClientesEnElGrupoEtario(((DatoVarietalPorGrupoEtario*) varietal)->mayoresDe50);
```

```
   std::cout << LINEA << std::endl;
}

//-----------------------------------------------Eliminar datos-----------------------------------------

/*
   PRE: Debe existir el ELEMENTO que deseo eliminar el cual debe ser un DatoVarietalPorGrupoEtario.
   POST: Elimino el dato del varietal.
 */
void eliminarDatosDeVarietales(ELEMENTO temp) {
   destruirDatoDeVarietal((DatoVarietalPorGrupoEtario*) temp);
}

//-----------------------------------------Funciones principal-----------------------------------------

void rankingVarietalesPorGrupoEtario(Lista* lMembresia, Lista* lUsuario, Lista* lCatalogos) {
   Lista* lVarietales = varietalesQueHay(lCatalogos);
   int iCantidad_Total_De_Ventas = 0;

   for (int i = 0; i < getCantidadDeElementosEnLaLista(lMembresia); i++) {
      ELEMENTO innerElemento;
      obtenerElementoDeLaLista(lMembresia, i, innerElemento);
      Lista* innerList = (Lista*) innerElemento;

      for (int x = 0; x < getCantidadDeElementosEnLaLista(innerList); x++) {
         ELEMENTO membresia;
         obtenerElementoDeLaLista(innerList, x, membresia);

         for (int j = 0; j < CANT_SELECCION; j++) {
```

```
            identificarVarietalDelVino((Membresia*) membresia, j, lUsuario, lCatalogos,
lVarietales);
            iCantidad_Total_De_Ventas++;
        }

    }
  }

  std::cout << std::endl << LINEA << std::endl << LINEA << "\n\t\t\t\tVarietal por
grupo etario menores de 30 años\n" << LINEA << std::endl << LINEA << std::endl;
  reordenarLista(lVarietales, compararMenoresDe30Anios, descendente);
  mostrarElementosDeLaLista(lVarietales, menoresDe30Anios);
  std::cout << std::endl << LINEA << std::endl << LINEA << "\n\t\t\t\tVarietal por
grupo etario entre 30 y 50 años\n" << LINEA << std::endl << LINEA << std::endl;
  reordenarLista(lVarietales, compararEntre30Y50Anios, descendente);
  mostrarElementosDeLaLista(lVarietales, entre30Y50Anios);
  std::cout << std::endl << LINEA << std::endl << LINEA << "\n\t\t\t\tVarietal por
grupo etario mayores de 50 años\n" << LINEA << std::endl << LINEA << std::endl;
  reordenarLista(lVarietales, compararMayoresDe50Anios, descendente);
  mostrarElementosDeLaLista(lVarietales, mayoresDe50Anios);

  std::cout << "\nCantidad total de ventas: " << iCantidad_Total_De_Ventas <<
std::endl;

  destruirListaYDatos(lVarietales, eliminarDatosDeVarietales);
}
```