```cpp
#include <iostream>
#include <string>

using namespace std;

int const CANTIDAD_CATEGORIAS = 200;
int const CANTIDAD_PRESTAMOS = 200;
int const CANTIDAD_PRESTATARIOS = 200;
string const SIN_ELEMENTOS = "No hay elementos para realizar esta operacion";

void limpiar_pantalla() {
  cout << string(50, '\n');
}

void capitalize(string &str) {
  for(unsigned int i = 0; i < str.size(); ++i) {
     str[i] = tolower(str[i]);
  }
  str[0] = toupper(str[0]);
}

struct Categoria {
  unsigned int codigo;
  string descripcion;
};

struct Lista_de_categorias {
  Categoria lista[CANTIDAD_CATEGORIAS];
  unsigned int longitud {0};
};

struct Prestatario {
  unsigned int codigo;
  string apellido;
  string nombre;
};

struct Lista_de_prestatarios {
  Prestatario lista[CANTIDAD_PRESTATARIOS];
  unsigned int longitud {0};
};

struct Prestamo {
  Categoria *categoria;
  Prestatario *prestatario;
```

```
  /*LOS CÓDIGOS QUE ASOCIAN A UNA CATEGORÍA Y A UN PRESTATARIO SON NÚMEROS ENTEROS.
  NO ES NECESARIO QUE SEAN STRUCTS COMPLETOS.
  ESTE COMENTARIO SE HIZO EN LA ENTREGA ANTERIOR
  */
  string descripcion;
  bool estado;
};

struct Lista_de_prestamos {
  Prestamo lista[CANTIDAD_PRESTAMOS];
  unsigned int longitud {0};
};

struct Almacen {
  Lista_de_categorias categorias;
  Lista_de_prestatarios prestatarios;
  Lista_de_prestamos prestamos;
};

struct Reporte {
  Categoria* categoria;
  /*LOS CÓDIGOS QUE ASOCIAN A UNA CATEGORÍA Y A UN REPORTE DEBERÍA SET NÚMEROS ENTEROS.
  NO ES NECESARIO QUE SEAN STRUCTS COMPLETOS.
  */
  int cantidad;
};

struct Lista_de_reporte {
  Reporte categorias[CANTIDAD_CATEGORIAS];
  int longitud {0};
};

struct Menu {
  int id;
  int cant_opciones;
  string opciones[7];
};

/*
PROPÓSITO: crear una categoria
PARÁMETROS:
  codigo: campo codigo de la categoria
  descripcion: el campo descripcion de categoria (precondicion: cadena no vacia)
RETORNO: una categoria
```

```
*/
Categoria crear_categoria(unsigned int codigo, string descripcion);
/*
PROPÓSITO: crear un prestamo
PARÁMETROS:
  categoria: puntero a una categoria
  prestatario: puntero a un prestatario
  descripcion: el campo descripcion del prestamo (precondicion: cadena no vacia)
RETORNO: un prestamo
*/
Prestamo crear_prestamo(Categoria &categoria, Prestatario &prestatario, string
descripcion);
/*
PROPÓSITO: crear un prestatario
PARÁMETROS:
  codigo: campo codigo de la categoria

SI ESTÁN CREANDO UN PRESTATARIO, ¿PARA QUÉ NECESITAN EL CÓDIGO DE
CATEGORÍA?
SEÑALADO EN LA CORRECCIÓN ANTERIOR
  nombre: el campo nombre del prestatario (precondicion: cadena no vacia)
  apellido: el campo apellido del prestatario (precondicion: cadena no vacia)
RETORNO: un prestatario
*/
Prestatario crear_prestatario(unsigned int codigo, string nombre, string apellido);
/*
PROPÓSITO: muestra en pantalla los datos de una categoria
PARÁMETROS:
  prestamo: muestra en pantall los datos de una categoria
*/
void mostrar_categoria(Categoria &categoria);
/*
PROPÓSITO: muestra en pantalla los datos de un prestatario
PARÁMETROS:
  prestamo: el prestatario a mostrar
*/
void mostrar_prestatario(Prestatario &prestatario);
/*
PROPÓSITO: muestra en pantalla los datos de un prestamo
PARÁMETROS:
  prestamo: el prestamo a mostrar
  */
void mostrar_prestamo(Prestamo &prestamo);
/*
PROPÓSITO: Verificar que no exista ningun prestamo con el codigo de la categoria dada
PARÁMETROS:
```

```
  categoria: la categoria a verificar
  prestamos: el almacen de prestamos
RETORNO: un booleano
*/
bool validar_eliminacion_categoria(Categoria &categoria, Lista_de_prestamos &prestamos);
/*
PROPÓSITO: Verificar que no exista ningun prestamo con el codigo del prestatario dado
PARÁMETROS:
  prestatario: el prestatario a verificar
  prestamos: el almacen de prestamos
RETORNO: un booleano
*/
bool validar_eliminacion_prestatario(Prestatario &prestatario, Lista_de_prestamos
&prestamos);
/*
PROPÓSITO: borrar una categoria
PARÁMETROS:
  categorias: la lista de categorias
  posicion: la posicion de la categoria a eliminar
*/
void borrar_categoria(Lista_de_categorias &categorias, unsigned int posicion);
/*
PROPÓSITO: borrar un prestatario
PARÁMETROS:
  prestatarios: la lista de prestatarios
  posicion: la posicion del prestario a eliminar
  */
void borrar_prestatario(Lista_de_prestatarios &prestatarios, unsigned int posicion);
/*
PROPÓSITO: borrar un prestamo
PARÁMETROS:
  pretamos: la lista de prestamos
  posicion: la posicion del prestamo a eliminar
*/
void borrar_prestamo(Lista_de_prestamos &prestamos, unsigned int posicion);
/*
PROPÓSITO: muestra una lista de categorias y pide al usuario que seleccione uno
PARÁMETROS:
  categorias: estructura donde se encuentra la informacion almacenada de categorias
RETORNO: un numero positivo
*/
unsigned int listar(Lista_de_categorias &categorias);
/*
PROPÓSITO: muestra una lista de prestatarios y pide al usuario que seleccione uno
PARÁMETROS:
  prestatarios: estructura donde se encuentra la informacion almacenada de prestatarios
```

```
RETORNO: un numero positivo
*/
unsigned int listar(Lista_de_prestatarios &prestatarios);
/*
PROPÓSITO: muestra una lista de prestamos y pide al usuario que seleccione uno
PARÁMETROS:
  prestamos: estructura donde se encuentra la informacion almacenada de prestamos
RETORNO: un numero positivo
*/
unsigned int listar(Lista_de_prestamos &prestamos);
/*
PROPÓSITO: Devuelve si existe algun prestamo para el prestatario dado
PARÁMETROS:
  prestamos: estructura donde se encuentra la informacion almacenada de prestamos
  prestatario: un prestatario para verificar si tiene algun prestamo
RETORNO: un booleano que indica si tiene o no prestamo
*/
bool tiene_prestamo(Lista_de_prestamos const &prestamos, Prestatario const &prestatario);
/*
PROPÓSITO: devuelve un prestamo
PARÁMETROS:
  prestamo: el prestamo a devolver
*/
void devolver_prestamo(Prestamo &prestamo);
/*
PROPÓSITO: almacena un prestamo
PARÁMETROS:
  prestamos: el almacen de prestamos
  prestamo: el prestamo a almacenar
*/
void almacenar_prestamo(Lista_de_prestamos &prestamos, Prestamo &prestamo);
/*
PROPÓSITO: almacena una categoria
PARÁMETROS:
  categorias: el almacen de categorias
  categoria: la categoria a almacenar
*/
void almacenar_categoria(Lista_de_categorias &categorias, Categoria &categoria);
/*
PROPÓSITO: almacena un prestatario
PARÁMETROS:
  pretatarios: el almacen de prestatarios
  prestatario: el prestatario a almacenar
*/
void almacenar_prestatario(Lista_de_prestatarios &prestatarios, Prestatario &prestatario);
/*
```

```
PROPÓSITO: obtener una categoria de la lista en la posicion dada
PARÁMETROS:
  categorias: el almacen de categorias
  posicion: la posicion de la categoria
*/
Categoria& pedir_categoria(Lista_de_categorias &categorias, unsigned int posicion);
/*
PROPÓSITO: obtener un prestatario de la lista en la posicion dada
PARÁMETROS:
  prestatarios: el almacen de prestatarios
  posicion: la posicion del prestatario
*/
Prestatario& pedir_prestatario(Lista_de_prestatarios &prestatarios, unsigned int posicion);
/*
PROPÓSITO: obtener un prestamo de la lista en la posicion dada
PARÁMETROS:
  prestamos: el almacen de prestamos
  posicion: la posicion del prestamo
*/
Prestamo& pedir_prestamo(Lista_de_prestamos &prestamos, unsigned int posicion);
/*
PROPÓSITO: muestra una lista de prestamos de un prestatario y pide al usuario que
seleccione uno
PARÁMETROS:
  prestamos: lista de prestamos
  prestatario: prestatario del cual seleccionar los prestamos
RETORNO: un puntero a un prestamo
*/
Prestamo& seleccionar_prestamo(Lista_de_prestamos &prestamos, Prestatario const
&prestatario);
/*
PROPÓSITO: generar un codigo automatico para la estructura a crear
PARÁMETROS:
  alamcen: estructura donde se encuentra la informacion almacenada
  almacen_especifico: el campo del almacen que se va a utilizar para el proposito
RETORNO: un numero positivo
*/
unsigned int obtener_codigo(Almacen &almacen, string almacen_especifico);
/*
PROPÓSITO: obtener una cadena de un campo de dato de la estructura correspondiente
PARÁMETROS:
  texto_a_mostrar: texto de referencia a lo que se debe ingresar
  requerido: bool que indica si este campo es o no obligatorio
RETORNO: una cadena
*/
string pedir_dato(string texto_a_mostrar);
```

```
/*
PROPÓSITO: busca un codigo de categoria en la lista de categorias y devuelve la posicion
PARÁMETROS:
  categorias: Lista de categorias
  codigo: el codigo de la categoria para verificar existencia
RETORNO: un entero que determina la posicion si existe, y si no, devuelve -1
*/
int existe_categoria(Lista_de_categorias &categorias, int codigo);
/*
PROPÓSITO: busca un codigo de prestatario en la lista de prestatarios y devuelve la
posicion
PARÁMETROS:
  prestatarios: Lista de prestatarios
  codigo: el codigo del prestatario para verificar existencia
RETORNO: un entero que determina la posicion si existe, y si no, devuelve -1
*/
int existe_prestatario(Lista_de_prestatarios &prestatarios, int codigo);
/*
PROPÓSITO: obtener una categoria existente, solicitando al usuario ingresar el codigo
PARÁMETROS:
  categorias: Lista de categorias para listar
  texto_a_mostrar: texto de referencia a lo que se debe ingresar
RETORNO: un unsigned int
//NO ES LO QUE RETORNA LA FUNCIÓN
*/
Categoria& pedir_categoria(Lista_de_categorias &categorias, string texto_a_mostrar);
/*
PROPÓSITO: obtener un prestatario existente, solicitando al usuario ingresar el codigo
PARÁMETROS:
  prestatarios: Lista de prestatarios para listar
  texto_a_mostrar: texto de referencia a lo que se debe ingresar
RETORNO: un unsigned int
//NO ES LO QUE RETORNA LA FUNCIÓN

*/
Prestatario& pedir_prestatario(Lista_de_prestatarios &prestatarios, string texto_a_mostrar);
/*
PROPÓSITO: dibujar en pantalla un reporte de cantidad de prestamos por cada categoria
PARÁMETROS:
  alamacen: el almacen donde se encuentra toda la informacion
RETORNO?
*/
Lista_de_reporte cantidad_prestamos_por_categoria(Almacen &almacen);
/*
PROPÓSITO: dibujar en pantalla un reporte de prestamos para una categoria dada
PARÁMETROS:
```

```
   prestamos: la lista de prestamos
   categoria: la categoria para filtrar prestamos
RETORNO?
*/
Lista_de_prestamos prestamos_por_categoria(Lista_de_prestamos &prestamos, Categoria
&categoria);
/*
PROPÓSITO: dibujar en pantalla un reporte de prestamos pendientes ordenados y
agrupados por el campo dado
PARÁMETROS:
   alamacen: el almacen donde se encuentra toda la informacion
   campo: texto C/c para categoria o P/p para prestatario
RETORNO?
*/
Lista_de_prestamos prestamos_pendientes_segun_criterio(Almacen &alamacen, string
campo);
/*
PROPÓSITO: dibujar los prestatarios que tienen prestamos pendientes
PARÁMETROS:
   alamacen: el almacen donde se encuentra toda la informacion
*/
void listar_prestatarios_con_prestamos_pendientes(Almacen &almacen);
/*
PROPÓSITO: muestra prestamos por pantalla
PARÁMETROS:
   prestamos: lista de prestamos
*/
void mostrar_prestamos(Lista_de_prestamos &prestamos);
/*
PROPÓSITO: muestra reporte de cantidad de prestamos por categoria por pantalla
PARÁMETROS:
   reportes: lista de reportes
*/
void mostrar_reportes(Lista_de_reporte reportes);
/*
PROPÓSITO: muestra un mensaje al usuario
PARÁMETROS:
   mensaje: el mensaje a mostrar
*/
void aviso(string mensaje);
/*
PROPÓSITO: Muestra un texto en pantalla y pide una opcion del tipo entero
PARÁMETROS:
   texto_a_mostrar: el texto que se debe mostrar antes de pedir el dato
RETORNO: un entero que es la opcion elegida
*/
```

```
int pedir_opcion(string texto_a_mostrar);
/*
PROPÓSITO: Muestra un texto en pantalla para pedir una confirmacino por si o no
PARÁMETROS:
  texto_a_mostrar: el texto que se debe mostrar antes de pedir confirmacion
RETORNO: devuelve un bool que indica si se confirma o no
*/
bool pedir_confirmacion(string texto_a_mostrar);

// FIXTURES
void crear_categorias_de_ejemplo(Almacen &almacen) {
  Categoria juegos = crear_categoria(0, "Juegos");
  almacenar_categoria(almacen.categorias, juegos);
  Categoria musica = crear_categoria(1, "Musica");
  almacenar_categoria(almacen.categorias, musica);
  Categoria libros = crear_categoria(2, "Libros");
  almacenar_categoria(almacen.categorias, libros);
}

void crear_prestatarios_de_ejemplo(Almacen &almacen) {
  Prestatario prestatario1 = crear_prestatario(0, "Garcia", "Petreco");
  almacenar_prestatario(almacen.prestatarios, prestatario1);
  Prestatario prestatario2 = crear_prestatario(1, "Lamuela", "Carlos");
  almacenar_prestatario(almacen.prestatarios, prestatario2);
  Prestatario prestatario3 = crear_prestatario(2, "Carrizo", "Osmar");
  almacenar_prestatario(almacen.prestatarios, prestatario3);
}

/*
PROPÓSITO: Dibuja un menu dado
PARÁMETROS:
  menu: un menu a dibujar
*/
void dibujar_menu(Menu &menu) {
  cout <<
"=============================================================================
===========" << endl;
  cout << "0 - Salir del programa" << endl;
  cout <<
"=============================================================================
===========" << endl;
  for(int i=0; i < menu.cant_opciones; i++) {
  cout << i+1 << " - " << menu.opciones[i] << endl;
  }
  if (menu.id != 0) {
```

```
    cout <<
"=============================================================================
===========" << endl;
    cout << "9 - Volver al menu principal" << endl;
    }
    cout <<
"=============================================================================
===========" << endl;
}


int main() {
    string mensaje;
    Almacen almacen = {};

    // DESCOMENTAR PARA CREAR PRESTATARIOS DE EJEMPLO
    crear_prestatarios_de_ejemplo(almacen);
    // DESCOMENTAR PARA CREAR CATEGORIAS DE EJEMPLO
    crear_categorias_de_ejemplo(almacen);
//EXCELENTE, NOS FACILITÓ LA CORRECCIÓN
    int opcion = -1;
    Menu menu_0 = {0, 3, {
        "Administrar y consultar Categorías y Prestatarios",
        "Administrar Préstamos",
        "Consultar Préstamos"}
    };
    Menu menu_1 = {1, 6, {
        "Agregar categoría",
        "Modificar categoría",
        "Eliminar categoría",
        "Agregar prestatario",
        "Modificar prestatario",
        "Eliminar prestatario"}
    };
    Menu menu_2 = {2, 4, {
        "Agregar préstamo",
        "Modificar préstamo",
        "Eliminar préstamo",
        "Devolver préstamo"}
    };
    Menu menu_3 = {3, 4, {
        "Cantidad de objetos prestados por categoría",
        "Listado de préstamos por categoría",
        "Listado de préstamos ordenados por categoría o prestatario",
        "Listar todos los prestatarios que tienen al menos un objeto prestado"}
    };
```

```
  Menu *menues[4]={&menu_0,&menu_1,&menu_2,&menu_3};
  int menu_actual = 0;

  cout << endl << endl <<"+========================= Administración de Préstamos
========================+" << endl;
  do {
     switch (menu_actual*10 + opcion) {
        case 11: {
           //agregar categoria
           string descripcion = pedir_dato("Ingrese la descripción de la categoría: ");
           unsigned int codigo = obtener_codigo(almacen, "categorias");
           Categoria categoria = crear_categoria(codigo, descripcion);
           almacenar_categoria(almacen.categorias, categoria);
           mensaje = "Categoria almacenada.";
           break;
        }
        case 12: {
           if (almacen.categorias.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
           //modificar categoria
           unsigned int posicion = listar(almacen.categorias);
           Categoria& seleccionada = pedir_categoria(almacen.categorias, posicion);
           string nueva_descripcion = pedir_dato("Ingrese la nueva descripción: ");
           capitalize(nueva_descripcion);
           seleccionada.descripcion = nueva_descripcion;
           mensaje = "La categoria fué modificada.";
           break;
        }
        case 13: {
           if (almacen.categorias.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
           // eliminar categoria
           unsigned int posicion = listar(almacen.categorias);
           Categoria& seleccionada = pedir_categoria(almacen.categorias, posicion);
           bool valido = validar_eliminacion_categoria(seleccionada, almacen.prestamos);
           if (valido) {
             borrar_categoria(almacen.categorias, posicion);
             mensaje = "Categoria eliminada";
           } else {
             mensaje = "La categoría no puede eliminarse debido a que hay préstamos
pendientes.";
           }
           break;
        }
        case 14:{
           //agregar prestatario
           string nombre = pedir_dato("Ingrese el nombre del prestatario: ");
           string apellido = pedir_dato("Ingrese el apellido del prestatario: ");
```

```
            unsigned int codigo = obtener_codigo(almacen, "prestatarios");
            Prestatario prestatario = crear_prestatario(codigo, nombre, apellido);
            almacenar_prestatario(almacen.prestatarios, prestatario);
            mensaje = "Prestatario almacenado.";
            break;
        }
        case 15:{
            if (almacen.prestatarios.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
            //modificar prestatario
            unsigned int posicion = listar(almacen.prestatarios);
            Prestatario& seleccionado = pedir_prestatario(almacen.prestatarios, posicion);
            string nuevo_nombre = pedir_dato("Ingrese el nuevo nombre: ");
            string nuevo_apellido = pedir_dato("Ingrese el nuevo apellido: ");
            capitalize(nuevo_nombre);
            capitalize(nuevo_apellido);
            seleccionado.nombre = nuevo_nombre;
            seleccionado.apellido = nuevo_apellido;
            mensaje = "Prestatario modificado.";
            break;
        }
        case 16: {
            if (almacen.prestatarios.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
            //eliminar prestatario
            unsigned int posicion = listar(almacen.prestatarios);
            Prestatario& seleccionado = pedir_prestatario(almacen.prestatarios, posicion);
            bool valido = validar_eliminacion_prestatario(seleccionado, almacen.prestamos);
            if (valido) {
              borrar_prestatario(almacen.prestatarios, posicion);
              mensaje = "Prestatario borrado.";
            } else {
              mensaje = "El prestatario no puede eliminarse debido a que hay préstamos
pendientes.";
            }
            break;
        }
        case 21: {
            if (almacen.categorias.longitud == 0 || almacen.prestatarios.longitud == 0) {
aviso(SIN_ELEMENTOS); break; }
            //agreagar prestamo
            Categoria& categoria = pedir_categoria(almacen.categorias, "Ingrese el codigo de
la categoria: ");
            Prestatario& prestatario = pedir_prestatario(almacen.prestatarios, "Ingrese el
codigo del prestatario: ");
            string descripcion = pedir_dato("Ingrese la descripcion del prestamo: ");
            Prestamo prestamo = crear_prestamo(categoria, prestatario, descripcion);
            almacenar_prestamo(almacen.prestamos, prestamo);
```

```
            mensaje = "Prestamo almacenado.";
            break;
        }
        case 22: {
            if (almacen.prestamos.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
            //modificar prestamo
            unsigned int posicion = listar(almacen.prestamos);
            Prestamo& seleccionado = pedir_prestamo(almacen.prestamos, posicion);
            string nueva_descripcion = pedir_dato("Ingrese la nueva descripción: ");
            capitalize(nueva_descripcion);
            seleccionado.descripcion = nueva_descripcion;
            mensaje = "Prestamo modificado.";
            break;
        }
        case 23: {
            if (almacen.prestamos.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
            //eliminar prestamo
            unsigned int posicion = listar(almacen.prestamos);
            borrar_prestamo(almacen.prestamos, posicion);
            mensaje = "El prestamo fue eliminado con exito.";
            break;
        }
        case 24: {
            if (almacen.prestamos.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
            //devolver prestamo

            unsigned int posicion = listar(almacen.prestatarios);
            Prestatario& prestatario = pedir_prestatario(almacen.prestatarios, posicion);
            if (tiene_prestamo(almacen.prestamos, prestatario)) {
                Prestamo& seleccionado = seleccionar_prestamo(almacen.prestamos,
prestatario);
                devolver_prestamo(seleccionado);
                mensaje = "El prestamo fue devuelto con exito";
            } else {
                mensaje = "No hay prestamos asociados al prestatario seleccionado";
            }
            break;
        }
        case 31: {
            if (almacen.prestamos.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
            //cant obj prestados por cat
            Lista_de_reporte reportes = cantidad_prestamos_por_categoria(almacen);
            mostrar_reportes(reportes);
            break;
        }
        case 32: {
```

```
            if (almacen.prestamos.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
            //list prestamo por cat
            unsigned int posicion = listar(almacen.categorias);
            Categoria& seleccionada = pedir_categoria(almacen.categorias, posicion);
            Lista_de_prestamos reporte = prestamos_por_categoria(almacen.prestamos,
seleccionada);
            mostrar_prestamos(reporte);
            break;
        }
        case 33: {
            if (almacen.prestamos.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
            //list ordenado prestamo por cat o prestatario
            string campo = pedir_dato("¿Clasificar el listado por Categoría (C) o Prestatario
(P)?");
            Lista_de_prestamos pendientes = prestamos_pendientes_segun_criterio(almacen,
campo);
            mostrar_prestamos(pendientes);
            break;
        }
        case 34: {
            if (almacen.prestamos.longitud == 0) { aviso(SIN_ELEMENTOS); break; }
            listar_prestatarios_con_prestamos_pendientes(almacen);
            break;
        }
        default: {
            limpiar_pantalla();
            if (opcion == 1 || opcion == 2 || opcion == 3)
                menu_actual = opcion;
            else
                menu_actual = 0;
        }
    }
    if (mensaje.size() > 0) {
        aviso(mensaje);
    }

    cout << endl;
    dibujar_menu(*menues[menu_actual]);
    opcion = pedir_opcion("[Ingrese una opción]: ");
    cout << "--------------------------------------------------------------------------------" << endl;
    cout << endl << endl;
    mensaje = "";
  } while (opcion != 0);

  return 0;
}
```

```
Categoria crear_categoria(unsigned int codigo, string descripcion) {
  capitalize(descripcion);
  Categoria categoria = {codigo, descripcion};
  return categoria;
}

Prestamo crear_prestamo(Categoria &categoria, Prestatario &prestatario, string descripcion)
{
  capitalize(descripcion);
  Prestamo prestamo = {&categoria, &prestatario, descripcion, true};
  return prestamo;
}

Prestatario crear_prestatario(unsigned int codigo, string nombre, string apellido) {
  capitalize(nombre);
  capitalize(apellido);
  Prestatario prestatario = {codigo, nombre, apellido};
  return prestatario;
}

void mostrar_categoria(Categoria &categoria) {
  cout << "CATEGORIA: " << categoria.codigo << " - " << categoria.descripcion << endl;
}

void mostrar_prestatario(Prestatario &prestatario) {
  cout << "PRESTATARIO: " << prestatario.codigo << " - " << prestatario.apellido << ", " <<
prestatario.nombre << endl;
}

void mostrar_prestamo(Prestamo &prestamo) {
  mostrar_categoria(*(prestamo.categoria));
  mostrar_prestatario(*(prestamo.prestatario));
  //REVISAR SI ESTO FUNCIONA (PRESTAMO.CATEGORIA Y
PRESTAMO.PRESTATARIO YA SON PUNTEROS).
  //CORRECCIÓN SEÑALADA ANTERIORMENTE

  cout << "DESCRIPCION: " << prestamo.descripcion << endl;
}

bool validar_eliminacion_categoria(Categoria &categoria, Lista_de_prestamos &prestamos)
{
  bool valido = true;
  for (unsigned int i = 0; i < prestamos.longitud; i++) {
      valido = &categoria != prestamos.lista[i].categoria;
      if (!valido) { break; }
```

```
  }
  return valido;
}

bool validar_eliminacion_prestatario(Prestatario &prestatario, Lista_de_prestamos
&prestamos) {
  bool valido = true;
  for (unsigned int i = 0; i < prestamos.longitud; i++) {
    valido = &prestatario != prestamos.lista[i].prestatario;
    if (!valido) { break; }
  }
  return valido;
}

void borrar_categoria(Lista_de_categorias &categorias, unsigned int posicion) {
  categorias.longitud--;
  //DEBERÍA EVITARSE EL CORRIMIENTO CUANDO EL ELEMENTO A BORRAR FUERA
EL ÚLTIMO
  //SUGERENCIA QUE SE HIZO EN LA CORRECCIÓN ANTERIOR

  for (unsigned int i = posicion; i < categorias.longitud; i++) {
    categorias.lista[i] = categorias.lista[i+1];
  }
}

void borrar_prestatario(Lista_de_prestatarios &prestatarios, unsigned int posicion) {
  prestatarios.longitud--;
  //DEBERÍA EVITARSE EL CORRIMIENTO CUANDO EL ELEMENTO A BORRAR FUERA
EL ÚLTIMO
  //SUGERENCIA QUE SE HIZO EN LA CORRECCIÓN ANTERIOR

  for (unsigned int i = posicion; i < prestatarios.longitud; i++) {
    prestatarios.lista[i] = prestatarios.lista[i+1];
  }
}

void borrar_prestamo(Lista_de_prestamos &prestamos, unsigned int posicion) {
  prestamos.longitud--;
  //DEBERÍA EVITARSE EL CORRIMIENTO CUANDO EL ELEMENTO A BORRAR FUERA
EL ÚLTIMO
  //SUGERENCIA QUE SE HIZO EN LA CORRECCIÓN ANTERIOR

  for (unsigned int i = posicion; i < prestamos.longitud; i++) {
    prestamos.lista[i] = prestamos.lista[i+1];
  }
}
```

```
unsigned int listar(Lista_de_categorias &categorias) {
   unsigned int longitud = categorias.longitud;
   unsigned int seleccion = 0;
   for (unsigned int i = 0; i < longitud; i++) {
      cout << i + 1 << " -> ";
      mostrar_categoria(categorias.lista[i]);
   }
   do {
      seleccion = pedir_opcion("[Seleccione un elmento de la lista]: ");
   } while (seleccion > longitud || seleccion == 0);
   return seleccion - 1;
//RETORNO, SIEMPRE RETORNA -1?
}

unsigned int listar(Lista_de_prestatarios &prestatarios) {
   int longitud = prestatarios.longitud;
   int seleccion = 0;
   for (int i = 0; i < longitud; i++) {
      cout << i + 1 << " -> ";
      mostrar_prestatario(prestatarios.lista[i]);
   }
   do {
      seleccion = pedir_opcion("[Seleccione un elmento de la lista]: ");
   } while (seleccion > longitud || seleccion == 0);
   return seleccion - 1;
//RETORNO, SIEMPRE RETORNA -1?
}

unsigned int listar(Lista_de_prestamos &prestamos) {
   int longitud = prestamos.longitud;
   int seleccion = 0;
   for (int i = 0; i < longitud; i++) {
      cout << i + 1 << " -> ";
      mostrar_prestamo(prestamos.lista[i]);
   }
   do {
      seleccion = pedir_opcion("[Seleccione un elmento de la lista]: ");
   } while (seleccion > longitud || seleccion == 0);
   return seleccion - 1;
//RETORNO, SIEMPRE RETORNA -1?
}

bool tiene_prestamo(Lista_de_prestamos const &prestamos, Prestatario const &prestatario)
{
   bool tiene = false;
```

```
    for (unsigned int i = 0; i < prestamos.longitud; i++) {
        tiene = &prestatario == prestamos.lista[i].prestatario;
        if (tiene) {
            break;
        }
    }
    return tiene;
}

void devolver_prestamo(Prestamo &prestamo){
    prestamo.estado = false;
}

void almacenar_prestamo(Lista_de_prestamos &prestamos, Prestamo &prestamo) {
    int longitud = prestamos.longitud;
    prestamos.lista[longitud] = prestamo;
    prestamos.longitud++;
}

void almacenar_categoria(Lista_de_categorias &categorias, Categoria &categoria) {
    int longitud = categorias.longitud;
    categorias.lista[longitud] = categoria;
    categorias.longitud++;
}

void almacenar_prestatario(Lista_de_prestatarios &prestatarios, Prestatario &prestatario) {
    int longitud = prestatarios.longitud;
    prestatarios.lista[longitud] = prestatario;
    prestatarios.longitud++;
}

Categoria& pedir_categoria(Lista_de_categorias &categorias, unsigned int posicion) {
    return categorias.lista[posicion];
}

Prestatario& pedir_prestatario(Lista_de_prestatarios &prestatarios, unsigned int posicion) {
    return prestatarios.lista[posicion];
}

Prestamo& pedir_prestamo(Lista_de_prestamos &prestamos, unsigned int posicion) {
    return prestamos.lista[posicion];
}

Prestamo& seleccionar_prestamo(Lista_de_prestamos &prestamos, Prestatario const
&prestatario) {
    int posiciones[CANTIDAD_PRESTAMOS];
```

```
    Lista_de_prestamos prestamos_seleccionados;
    int j = 0;
    for (unsigned int i = 0; i < prestamos.longitud; i++) {
        if (&prestatario == prestamos.lista[i].prestatario) {
            posiciones[j] = i;
            j++;
        }
    }
    int posicion = listar(prestamos_seleccionados);
    return pedir_prestamo(prestamos, posiciones[posicion]);
}

unsigned int obtener_codigo(Almacen &almacen, string almacen_especifico) {
    unsigned int codigo = 0;
    if (almacen_especifico == "categorias") {
        int longitud = almacen.categorias.longitud;
        if (longitud > 0) {
            Categoria ultima = pedir_categoria(almacen.categorias, longitud - 1);
            codigo = ultima.codigo + 1;
        }
    }
    if (almacen_especifico == "prestatarios") {
        int longitud = almacen.prestatarios.longitud;
        if (longitud > 0) {
            Prestatario ultimo = pedir_prestatario(almacen.prestatarios, longitud - 1);
            codigo = ultimo.codigo + 1;
        }
    }
    return codigo;
}

string pedir_dato(string texto_a_mostrar) {
    string cadena;
    cout << texto_a_mostrar;
    getline(cin, cadena);
    return cadena;
}

int existe_categoria(Lista_de_categorias &categorias, int codigo) {
    int posicion = -1;
    for (unsigned int i = 0; i < categorias.longitud; i++) {
        if (categorias.lista[i].codigo == codigo) {
            posicion = i;
            break;
        }
    }
```

```
    return posicion;
}

int existe_prestatario(Lista_de_prestatarios &prestatarios, int codigo) {
    int posicion = -1;
    for (unsigned int i = 0; i < prestatarios.longitud; i++) {
        if (prestatarios.lista[i].codigo == codigo) {
            posicion = i;
            break;
        }
    }
    return posicion;
}

Categoria& pedir_categoria(Lista_de_categorias &categorias, string texto_a_mostrar) {
    bool confirmacion = pedir_confirmacion("Desea listar las categorias existentes? [s/n]: ");
    int posicion;
    if (confirmacion) {
        posicion = listar(categorias);
    } else {
        do {
            int codigo = pedir_opcion(texto_a_mostrar);
            posicion = existe_categoria(categorias, codigo);
        } while (posicion == -1);
    }
    return pedir_categoria(categorias, posicion);
}

Prestatario& pedir_prestatario(Lista_de_prestatarios &prestatarios, string texto_a_mostrar) {
    bool confirmacion = pedir_confirmacion("Desea listar las prestatarios existentes? [s/n]: ");
    int posicion;
    if (confirmacion) {
        posicion = listar(prestatarios);
    } else {
        do {
            int codigo = pedir_opcion(texto_a_mostrar);
            posicion = existe_prestatario(prestatarios, codigo);
        } while (posicion == -1);
    }
    return pedir_prestatario(prestatarios, posicion);
}

Lista_de_reporte cantidad_prestamos_por_categoria(Almacen &almacen) {
    /*
    * ciclar la lista de categorias y llamar a una funcion auxiliar
    * que retorna la lista de prestamos dada una categoria
```

```
   * y mostrar la longitud de esta y se dibuja en pantalla
   * Esta funcion recorre la lista de prestamos buscando por aquellas que tengan en el
campo
   * categoria el codigo de la categoria recibida por parametro y arma el reporte.
   */
   Lista_de_reporte reportes;
   return reportes;
}

Lista_de_prestamos prestamos_por_categoria(Lista_de_prestamos &prestamos, Categoria
&categoria) {
   /*
   * llama una funcion auxiliar que retorna una lista de prestamos dada una categoria.
   * Esta funcion recorre la lista de prestamos buscando por aquellas que tengan en el
campo
   * categoria el codigo de la categoria recibida por parametro y las pone en un array que
devuelve.
   */
   Lista_de_prestamos prestamos_filtrados;
   return prestamos_filtrados;
}

Lista_de_prestamos prestamos_pendientes_segun_criterio(Almacen &alamacen, string
campo) {
   /*
   * revisa campo si es categoria o prestatario, para luego recorrer los prestamos pendientes
   * y quedarse con el codigo de categoria o prestatario segun corresponda.
   * Se filtra el array para eliminar repeticiones y ordenarlo para luego retornarlo.
   */
   Lista_de_prestamos prestamos;
   return prestamos;
}

void listar_prestatarios_con_prestamos_pendientes(Almacen &almacen) {
   /*
   * Se recorren los prestamos y si el campo estado es true se alamacena
   * el prestatario en un array auxiliar para luego mostrarlo por pantalla
   */
}

void mostrar_prestamos(Lista_de_prestamos &prestamos) {
   /*
   * cicla prestamos y muestra cada uno por pantalla
   */
}
```

```
void mostrar_reportes(Lista_de_reporte reportes) {
  /*
  * cicla reportes y muestra cada uno por pantalla
  */
}

void aviso(string mensaje) {
  limpiar_pantalla();
  cout << mensaje << endl;
}

int pedir_opcion(string texto_a_mostrar){
  bool condicion;
  int n;
  do {
     cout << texto_a_mostrar;
     cin >> n;
     condicion = cin.fail();
     cin.clear();
     cin.ignore(INT_MAX, '\n');
  } while(condicion);
  return n;
}

bool pedir_confirmacion(string texto_a_mostrar){
  bool condicion;
  string confirmacion;
  do {
     cout << texto_a_mostrar;
     confirmacion = cin.get();
     condicion = cin.fail();
     if (!condicion) {
        confirmacion = tolower(confirmacion[0]);
        condicion = confirmacion == "s" || confirmacion == "n";
     }
     cin.clear();
     cin.ignore(256, '\n');
  } while(!condicion);
  return confirmacion == "s";
}
```

ERRORES OBSERVADOS

- Muchas observaciones señaladas en la entrega anterior no fueron subsanadas, volvimos a marcarlas. Por favor trabajar sobre este punto.

- Sólo encontramos un error en la opción para devolver préstamos, quizá podrían ver por qué en la función “lista()” retorna siempre el valor -1.

OTRAS OBSERVACIONES Y SUGERENCIAS:

- Se destacan las funciones: crear_prestatarios_de_ejemplo() y crear_categorias_de_ejemplo(), ya que nos facilitó mucho la corrección.

Calificación: por todo lo expuesto anteriormente, la calificación es Aprobado.