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

using namespace std;

int const CANTIDAD_CATEGORIAS = 200;
int const CANTIDAD_PRESTAMOS = 200;
int const CANTIDAD_PRESTATARIOS = 200;

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.
```
  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;
};
```
Buen agrupamiento de los arreglos con sus respectivas dimensiones lógicas.

```
/*
PROPÃ“SITO: crear una categoria
PARÃMETROS:
    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) {
    /*
     *  crea una variable del tipo Categoria utilizando la descripcion y el codigo pasados por
parametro
    */
     Categoria categoria = {0, "una categoria"};
     return categoria;
   }

/*
PROPÃ“SITO: crear un prestamo
PARÃMETROS:
    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)
{
  /*
    *  crea una variable del tipo Prestamo utilizando la descripcion, categoria, prestamo y el
codigo pasados por parametro
  */
  Categoria cat = {0, "una categoria"};
  Prestatario prest = {0, "apellido", "nombre"};
  Prestamo prestamo = {&cat, &prest, "un prestamo", true};
  return prestamo;
}

/*
PROPÃ“SITO: crear un prestatario
PARÃMETROS:
    codigo: campo codigo de la categoria
```
Si están creando un prestatario, ¿para qué necesitan el código de categoría?

```
    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) {
  /*
   *  crea una variable del tipo Prestatario utilizando el nombre, apellido y el codigo pasados
por parametro
  */
  Prestatario prestatario = {0, "nombre", "apellido"};  ¿Por qué el código es 0 y los otros
argumentos son strings literales en vez de usar los parámetros recibidos en la función?
  return prestatario;
}

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

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

Falta documentación de esta función
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).
  cout << "DESCRIPCION: " << prestamo.descripcion << endl;
}

/*
PROPÃ“SITO: Verificar que no exista ningun prestamo con el codigo de la categoria dada
PARÃMETROS:
    categoria: la categoria a verificar
    prestamos: el almacen de prestamos
RETORNO: un booleano
*/
bool validar_eliminacion_categoria(Categoria &categoria, Lista_de_prestamos &prestamos)
{
  /*
   * Recorre y busca el codigo de la categoria dada en la lista de prestamos
   * si lo encuentra devuelve false, sino devuelve true
   */
```

```
  return true;
 }

/*
PROPÃ“SITO: Verificar que no exista ningun prestamo con el codigo del prestatario dado
PARÃMETROS:
    prestatario: el prestatario a verificar
    prestamos: el almacen de prestamos
RETORNO: un booleano
*/
bool validar_eliminacion_prestatario(Prestatario &prestatario, Lista_de_prestamos
&prestamos) {
  /*
   * Recorre y busca el codigo del prestatario dado en la lista de prestamos
   * si lo encuentra devuelve false, sino devuelve true
   */
  return true;
 }

/*
PROPÃ“SITO: borrar una categoria
PARÃMETROS:
    categorias: la lista de categorias
    posicion: la posicion de la categoria a eliminar
*/
void borrar_categoria(Lista_de_categorias &categorias, unsigned int posicion) {
  /*
   * itera sobre la lista de categorias buscando la posicion dada
   * cuando la encuentra la pisa realizando un corrimiento de ser necesario, si no la ignora
   * y decrementa en una unidad a la longitud de la lista de categorias
   */
¿Cuándo no sería necesario el corrimiento? Si es cuando el elemento a eliminar es el
último, es correcto.
}

/*
PROPÃ“SITO: borrar un prestatario
PARÃMETROS:
    prestatarios: la lista de prestatarios
    posicion: la posicion del prestario a eliminar
*/
void borrar_prestatario(Lista_de_prestatarios &prestatarios, unsigned int posicion) {
  /*
   * itera sobre la lista de prestatarios buscando la posicion dada
   * cuando la encuentra la pisa realizando un corrimiento de ser necesario, si no la ignora
   * y decrementa en una unidad a la longitud de la lista de prestatarios
```

```
   */
}

/*
PROPÃ“SITO: borrar un prestamo
PARÃMETROS:
    pretamos: la lista de prestamos
    posicion: la posicion del prestamo a eliminar
*/
void borrar_prestamo(Lista_de_prestamos &prestamos, unsigned int posicion) {
  /*
   * itera sobre la lista de prestamos buscando la posicion dada
   * cuando la encuentra la pisa realizando un corrimiento de ser necesario, si no la ignora
   * y decrementa en una unidad a la longitud de la lista de prestamos
   */
}

/*
PROPÃ“SITO: generar un codigo automatico para la estructura a crear
PARÃMETROS:
    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 &almancen, string almacen_especifico) {
  /*
   * segun el almacen_especifico buscar la lista en el almacen y pedirle la longitud y restarle
1
   */
   unsigned int codigo = 0;
   return codigo;
}

/*
PROPÃ“SITO: muestra una lista de categorias/prestamos/prestatarios y pide al usuario que
seleccione uno
PARÃMETROS:
    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 listar(Almacen &almacen, string almacen_especifico) {
  /*
   * segun el alamacen_especifico se recorre y muestra con una estructura de bucle la lista
de esta entidad
   * el usuario ingresa el orden y retorna el numero de orden - 1
```

```
  */
   return 0;
 }

/*
PROPÃ“SITO: muestra una lista de prestamos de un prestatario y pide al usuario que
seleccione uno
PARÃMETROS:
    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
&prestatario) {
  /*
   * se recorre y muestra con una estructura de bucle la lista de prestamos de un prestatario
   * seleccionado y se le pide al usuario que eliga uno de las opciones listadas
   * el usuario ingresa la seleccion y se valida que sea correcta y lo devuelve
  */
  return &prestamos.lista[0];
}

/*
PROPÃ“SITO: devuelve un prestamo
PARÃMETROS:
    prestamo: el prestamo a devolver
*/
void devolver_prestamo(Prestamo &prestamo){
  /*
   * Cambia el campo estado de True a False
   */
La devolución debe mostrar los prestatarios y categorías, tal como lo solicita el enunciado.
}

/*
PROPÃ“SITO: almacena un prestamo
PARÃMETROS:
    prestamos: el almacen de prestamos
    prestamo: el prestamo a almacenar
*/
void almacenar_prestamo(Lista_de_prestamos &prestamos, Prestamo &prestamo) {
  /*
   * agrega un nuevo prestamo en la lista donde se almacenan los prestamos
   * Ademas mantiene la longitud de la lista de prestamos
  */
  prestamos.lista[0] = prestamo;
```

```
}

/*
PROPÃ“SITO: almacena una categoria
PARÃMETROS:
    categorias: el almacen de categorias
    categoria: la categoria a almacenar
*/
void almacenar_categoria(Lista_de_categorias &categorias, Categoria &categoria) {
  /*
   * agrega una nueva categoria en la lista donde se almacenan las categorias
   * Ademas mantiene la longitud de la lista de categorias
   */
  categorias.lista[0] = categoria;
}

/*
PROPÃ“SITO: almacena un prestatario
PARÃMETROS:
    pretatarios: el almacen de prestatarios
    prestatario: el prestatario a almacenar
*/
void almacenar_prestatario(Lista_de_prestatarios &prestatarios, Prestatario &prestatario) {
  /*
   * agrega un nuevo prestatario en la lista donde se almacenan los prestatarios
   * Ademas mantiene la longitud de la lista de prestatarios
   */
  prestatarios.lista[0] = prestatario;
}

/*
PROPÃ“SITO: obtener una categoria de la lista en la posicion dada
PARÃMETROS:
    categorias: el almacen de categorias
    posicion: la posicion de la categoria
*/
Categoria* pedir_categoria(Lista_de_categorias &categorias, unsigned int posicion) {
  /*
   * recorrer la lista de categorias hasta la posicion dada y devolver esa posicion de memoria
   */
  return &categorias.lista[0];
}

/*
PROPÃ“SITO: obtener un prestatario de la lista en la posicion dada
PARÃMETROS:
```

```
    prestatarios: el almacen de prestatarios
    posicion: la posicion del prestatario
*/
Prestatario* pedir_prestatario(Lista_de_prestatarios &prestatarios, unsigned int posicion) {
  /*
   * recorrer el lista de prestatarios hasta la posicion dada y devolver esa posicion de
memoria
   */
  return &prestatarios.lista[0];
}

/*
PROPÃ“SITO: obtener un prestamo de la lista en la posicion dada
PARÃMETROS:
    prestamos: el almacen de prestamos
    posicion: la posicion del prestamo
*/
Prestamo* pedir_prestamo(Lista_de_prestamos &prestamos, unsigned int posicion) {
  /*
   * recorrer la lista de prestamos hasta la posicion dada y devolver esa posicion de memoria
  */
  return &prestamos.lista[0];
}

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

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;
}

/*
PROPÃ“SITO: obtener una cadena de un campo de dato de la estructura correspondiente
PARÃMETROS:
    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  ¿Retorna la misma cadena que se ingresó?
*/
string pedir_dato(string texto_a_mostrar, bool requerido) {
  /*
   * solicita al usuario el ingreso de una cadena por teclado,
   * verifica si es obligatoria o no segun el parametro 'requerido', volviendo a solicitarlo de ser
necesario
   * luego de la validacion retorna la cadena ingresada
  */
  return "una cadena";
}

/*
PROPÃ“SITO: obtener una categoria existente, solicitando al usuario ingresar el codigo
PARÃMETROS:
    categorias: Lista de categorias para listar
    texto_a_mostrar: texto de referencia a lo que se debe ingresar
    requerido: bool que indica si este campo es o no obligatorio
RETORNO: un unsigned int
*/
Categoria* pedir_categoria(Lista_de_categorias &categorias, string texto_a_mostrar) {
  /*
   * Pide listar las categorias existentes, si el usuario indica que si
   * muestra el texto_a_mostrar en la pantalla y recibe un codigo
   * se valida la existencia del codigo
   * si no es valido se vuelve a repetir el proceso
  */
  return &categorias.lista[0];
}

/*
PROPÃ“SITO: obtener un prestatario existente, solicitando al usuario ingresar el codigo
PARÃMETROS:
    prestatarios: Lista de prestatarios para listar
    texto_a_mostrar: texto de referencia a lo que se debe ingresar
```

```
    requerido: bool que indica si este campo es o no obligatorio
RETORNO: un unsigned int
*/
Prestatario* pedir_prestatario(Lista_de_prestatarios &prestatarios, string texto_a_mostrar) {
  /*
   * Pide listar los prestatarios existentes, si el usuario indica que si
   * muestra el texto_a_mostrar en la pantalla y recibe un codigo
   * se valida la existencia del codigo
   * si no es valido se vuelve a repetir el proceso
  */
  return &prestatarios.lista[0];
}

struct Reporte {
  Categoria* categoria;
  int cantidad;
};

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

/*
PROPÃ“SITO: dibujar en pantalla un reporte de cantidad de prestamos por cada categoria
PARÃMETROS:
    alamacen: el almacen donde se encuentra toda la informacion
*/
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;
}

/*
PROPÃ“SITO: dibujar en pantalla un reporte de prestamos para una categoria dada
PARÃMETROS:
    prestamos: la lista de prestamos
    categoria: la categoria para filtrar prestamos
```

```
*/
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;
}

/*
PROPÃ“SITO: dibujar en pantalla un reporte de prestamos pendientes ordenados y
agrupados por el campo dado
PARÃMETROS:
    alamacen: el almacen donde se encuentra toda la informacion
    campo: texto C/c para categoria o P/p para prestatario
*/
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;
}

/*
PROPÃ“SITO: dibujar los prestatarios que tienen prestamos pendientes
PARÃMETROS:
    alamacen: el almacen donde se encuentra toda la informacion
*/
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
   */
}

/*
PROPÃ“SITO: muestra prestamos por pantalla
```

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

/*
PROPÃ“SITO: muestra reporte de cantidad de prestamos por categoria por pantalla
PARÃMETROS:
    reportes: lista de reportes
*/
void mostrar_reportes(Lista_de_reporte reportes) {
  /*
   * cicla reportes y muestra cada uno por pantalla
  */
}

/*
PROPÃ“SITO: muestra un mensaje al usuario
PARÃMETROS:
    mensaje: el mensaje a mostrar
*/
void aviso(string mensaje) {
  cout << mensaje << endl;
}
```

Falta documentación de esta función

```
int main() {
  Almacen almacen = {};

  int opcion = -1;
  Menu menu_0 = {0, 3, {
    "Administrar y consultar CategorìAs y Prestatarios",
    "Administrar Preìstamos",
    "Consultar Preìstamos"}
  };
  Menu menu_1 = {1, 6, {
    "Agregar categorìA",
    "Modificar categorìA",
    "Eliminar categorìA",
    "Agregar prestatario",
    "Modificar prestatario",
```

```
    "Eliminar prestatario"}
  };
  Menu menu_2 = {2, 4, {
    "Agregar prelstamo",
    "Modificar prÃ©stamo",
    "Eliminar prÃ©stamo",
    "Devolver prÃ©stamo"}
  };
  Menu menu_3 = {3, 4, {
    "Cantidad de objetos prestados por categorìa",
    "Listado de prelstamos por categorìa",
    "Listado de prelstamos ordenados por categorìa 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 <<"+========================= AdministraciÃ³n de
PrÃ©stamos ========================+" << endl;
  do {
¿Por qué tanta complejidad para un menú?
    switch (menu_actual*10 + opcion) {
case 11: {
        //agregar categoria
        string descripcion = pedir_dato("Ingrese la descripciÃ³n de la categorÃa: ", true);
        unsigned int codigo = obtener_codigo(almacen, "categorias");
        Categoria categoria = crear_categoria(codigo, descripcion);
        almacenar_categoria(almacen.categorias, categoria);
        mostrar_categoria(categoria);
        break;
      }
      case 12: {
        //modificar categoria
        unsigned int posicion = listar(almacen, "categorias");
        Categoria *seleccionada = pedir_categoria(almacen.categorias, posicion);
        string nueva_descripcion = pedir_dato("Ingrese la nueva descripciÃ³n: ", true);
        (*seleccionada).descripcion = nueva_descripcion;
        mostrar_categoria(*seleccionada);
        break;
      }
      case 13: {
        // 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);
```

```
      } else {
        aviso("La categorìa no puede eliminarse debido a que hay prelstamos pendientes.");
      }
      break;
    }
    case 14:{
      //agregar prestatario
      string nombre = pedir_dato("Ingrese el nombre del prestatario: ", true);
      string apellido = pedir_dato("Ingrese el apellido del prestatario: ", true);
      unsigned int codigo = obtener_codigo(almacen, "prestatario");
      Prestatario prestatario = crear_prestatario(codigo, nombre, apellido);
      almacenar_prestatario(almacen.prestatarios, prestatario);
      mostrar_prestatario(prestatario);
      break;
    }
    case 15:{
      //modificar prestatario
      unsigned int posicion = listar(almacen, "prestatarios");
      Prestatario *seleccionado = pedir_prestatario(almacen.prestatarios, posicion);
      string nuevo_nombre = pedir_dato("Ingrese el nuevo nombre: ", true);
      string nuevo_apellido = pedir_dato("Ingrese el nuevo apellido: ", true);
      (*seleccionado).nombre = nuevo_nombre;
      (*seleccionado).apellido = nuevo_apellido;
      mostrar_prestatario(*seleccionado);
      break;
    }
    case 16: {
      //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);
      } else {
        aviso("El prestatario no puede eliminarse debido a que hay prÃ©stamos
pendientes.");
      }
      break;
    }
    case 21: {
      //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: ", true);
```

```
      Prestamo prestamo = crear_prestamo(categoria, prestatario, descripcion);
      almacenar_prestamo(almacen.prestamos, prestamo);
      mostrar_prestamo(prestamo);
      break;
    }
    case 22: {
      //modificar prestamo
      unsigned int posicion = listar(almacen, "prestamo");
      Prestamo *seleccionado = pedir_prestamo(almacen.prestamos, posicion);
      string nueva_descripcion = pedir_dato("Ingrese la nueva descripciÃ³n: ", true);
      (*seleccionado).descripcion = nueva_descripcion;
      mostrar_prestamo(*seleccionado);
      break;
    }
    case 23: {
      //eliminar prestamo
      unsigned int posicion = listar(almacen, "prestamos");
      Prestamo *seleccionado = pedir_prestamo(almacen.prestamos, posicion);
      borrar_prestamo(almacen.prestamos, posicion);
      aviso("El prestamo fue eliminado con exito.");
      break;
    }
    case 24: {
      //devolver prestamo
      unsigned int posicion = listar(almacen, "prestatario");
      Prestatario *prestatario = pedir_prestatario(almacen.prestatarios, posicion);
      Prestamo *seleccionado = seleccionar_prestamo(almacen.prestamos, *prestatario);
      devolver_prestamo(*seleccionado);
      aviso("El prestamo fue devuelto con exito");
      break;
    }
    case 31: {
      //cant obj prestados por cat
      Lista_de_reporte reportes = cantidad_prestamos_por_categoria(almacen);
      mostrar_reportes(reportes);
      break;
    }
    case 32: {
      //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: {
        //list ordenado prestamo por cat o prestatario
        string campo = pedir_dato("Â¿Clasificar el listado por CategorÃa (C) o Prestatario
(P)?", true);
        Lista_de_prestamos pendientes = prestamos_pendientes_segun_criterio(almacen,
campo);
        mostrar_prestamos(pendientes);
        break;
      }
      case 34: {
        //list prestatario la menos un obj prestado
        listar_prestatarios_con_prestamos_pendientes(almacen);
        break;
      }
```

Buena modularización en cada case de menú

```
      default:
      if (opcion == 1 || opcion == 2 || opcion == 3)
        menu_actual = opcion;
      else
        menu_actual = 0;
    }
    dibujar_menu(*menues[menu_actual]);
    cout << "[Ingrese una opciÃ³n]: ";
    cin >> opcion;
    cout << "--------------------------------------------------------------------------------" << endl;
    cout << endl << endl;
  } while (opcion != 0);

  return 0;
}
```

ERRORES OBSERVADOS

- Las opciones 1 y 2 de la opción 2 del menú principal provocan error.
- El valor de retorno, en la documentación, debe explicar qué representa o para qué sirve.
- Muchas funciones están codificadas en lugar de explicar su algoritmo (tal como solicita el enunciado).

OTRAS OBSERVACIONES Y SUGERENCIAS:

- El código es muy complejo en cuestiones que podrían resolverse de manera más simple. Se considera buena práctica de programación que el código sea lo más legible y claro posible, para permitir a cualquier programador comprender lo que se hizo, especialmente cuando se trabaja en equipo. (http://docforge.com/wiki/Best_practices).
- En general, la modularización es muy buena (permite una buena reutilización de las funciones).
- El struct préstamo hace referencia a structs completos en el código categoría y el código de prestatario. Estos punteros podrían ocasionar problemas si el programa se guarda en disco y no en memoria (aunque no es tema visto en la materia, por lo que se considera correcto en la entrega, mientras luego se codifique adecuadamente).
- Puede ser útil realizar una función que reciba por parámetro un string y retorne el mismo string con su primera letra en mayúscula y el resto en minúsculas. Esta operación debe realizarse varias veces durante el programa (cada vez que el usuario ingresa el nombre o el apellido de un prestatario).

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