Escuela Politécnica Nacional

Documentación del Código

Check IN/OUT de un Hotel en C++ con NFC

Integrantes

- Jahir Rocha
- Sara Rosero
- Kevin Tacuri

Documentación del código

A continuación, se presenta la documentación del código C++ proporcionado:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <unistd.h>
#include <cctype>
#include <windows.h>
#include <vector>
using namespace std;
```

Este fragmento de código incluye las bibliotecas necesarias para el programa. Aquí hay una breve descripción de algunas de las bibliotecas incluidas:

- iostream: Biblioteca estándar de C++ para entrada y salida.
- fstream: Biblioteca para manejar archivos de texto.
- iomanip: Biblioteca para formatear la salida en pantalla.
- unistd.h: Biblioteca específica de Unix para funciones relacionadas con el sistema operativo.
- cctype: Biblioteca para funciones relacionadas con caracteres.
- windows.h: Biblioteca para funciones específicas de Windows.
- vector: Biblioteca para usar vectores, que son arreglos dinámicos.

```
const string LIMPIAR = "\x1B[2J\x1B[H";
const string ARCHIVORESERVACIONES = "../data/reservacion.txt";
const string ARCHIVOCLIENTESGENERAL = "../data/clienteGeneral.txt";
const string ARCHIVOTAG = "../data/TAG.txt";
```

Estas son constantes que contienen rutas de archivos y secuencias de escape para el formateo de la salida. LIMPIAR se usa para limpiar la pantalla, mientras que las otras tres constantes almacenan las rutas de archivos que se utilizarán más adelante en el programa.

```
struct Cliente
{
    int ID;
    string Cedula;
    string Nombre;
    string Direccion;
    int Asistencia;
};
```

Aquí se define una estructura llamada Cliente que representa a un cliente. Contiene los siguientes campos: ID (identificación numérica única), Cedula (número de cédula del cliente), Nombre, Direccion y Asistencia (un

contador de asistencias).

```
struct Reservacion
{
    int ID_Cliente;
    string Nombre;
    int Dias;
    string TAG;
    string Check;
};
```

Se define otra estructura llamada Reservacion que representa una reservación. Tiene campos como ID_Cliente (que se relaciona con el cliente), Nombre (nombre de la reservación), Dias (duración de la reservación), TAG y Check (estado de la reservación).

```
vector<Cliente> clientes;
vector<Reservacion> reservaciones;
```

Se declaran dos vectores para almacenar múltiples instancias de clientes y reservaciones, respectivamente.

```
enum color {azul=1, verde, turqueza, rojo, rosa, amarillo, blanco, morado};
```

Se define un enumerador color que asigna valores a diferentes colores que se pueden utilizar para dar formato al texto en la salida.

```
/**
 * PonerColor: Funcion para dar color al texto
 * @param c: color a dar
*/
string prjPonerColor(color c)
{
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), c);
    return "";
}
```

Se define una función prjPonerColor que se utiliza para cambiar el color del texto en la consola. Toma un parámetro c que es un valor de color según el enumerador color.

```
/**
 * DividirCadena: Funcion para dividir una cadena en un vector
 * @param str: cadena a dividir
 * @param delim: caracter a dividir
*/
```

```
vector<string> prjDivirCadena(const string str, const char delim)
{
    vector<string> v;
    string strDiv;
    stringstream s(str);
    while (getline(s, strDiv, delim))
        v.push_back(strDiv);
    return v;
}
```

Esta función prjDivirCadena se utiliza para dividir una cadena en varias partes utilizando un delimitador. Toma una cadena str y un carácter delimitador delim, y devuelve un vector de las partes divididas de la cadena.

```
/**
 * validarNumero: Función para validar si un número es positivo
 * @param numero: número a validar
 * @return true si el número es positivo, false en caso contrario
 */
bool prjValidarNumero(string numero)
{
    for (int i = 0; i < numero.length(); i++)
        if (!isdigit(numero[i]))
            return false;
    return true;
}
```

La función prjValidarNumero se utiliza para verificar si una cadena de caracteres representa un número positivo. Recorre cada carácter de la cadena y verifica si es un dígito. Si todos los caracteres son dígitos, devuelve true; de lo contrario, devuelve false.

```
/**
 * ObtenerNumeroPositivo: Función para obtener un número positivo desde la entrada
estándar
 * @param label: etiqueta del número
 * @param numero: número a obtener (por referencia)
 */
void prjObtenerNumeroPositivo(const string label, int &numero)
{
    string str;
    bool esNumeroValido = false;

    do
    {
        cout << label;
        cin >> str;
        esNumeroValido = prjValidarNumero(str);
        if (!esNumeroValido)
        {
            cout << "Entrada no válida. Introduzca un número positivo" << endl;
```

```
        }
    } while (!esNumeroValido);

    numero = stoi(str);
}
```

La función prjObtenerNumeroPositivo se utiliza para obtener un número positivo desde la entrada estándar. Toma una etiqueta label que se muestra al usuario para indicar qué número debe ingresar. Se repite hasta que el usuario ingrese un número válido (positivo) y luego almacena ese número en la variable numero utilizando una referencia.

```
/**
 * ValidarIdClientes: Función para validar si un ID de cliente es único
 * @param id: ID a validar
 * @param clientes: vector de clientes
 * @return true si el ID es único, false en caso contrario
 */
bool prjValidarIDClientes(int id, const vector<Cliente> &clientes)
{
    for (const auto &cliente : clientes) {
        if (cliente.ID == id) {
            return false;
        }
    }
    return true;
}
```

La función prjValidarIDClientes se utiliza para verificar si un ID de cliente es único dentro del vector de clientes. Recorre el vector y compara el ID proporcionado con los IDs existentes en el vector. Devuelve true si el ID es único y false si ya existe.

```
/**
 * ValidarIDReservacion: Función para validar si un ID de reservación es único
 * @param id: ID a validar
 * @param reservaciones: vector de reservaciones
 * @return true si el ID es único, false en caso contrario
 */
bool prjValidarIDReservacion(int id, const vector<Reservacion> &reservaciones)
{
    for (const auto &reservacion : reservaciones)
    {
        if (reservacion.ID_Cliente == id) {
            return false;
        }
    }
    return true;
}
```

La función prjValidarIDReservacion se utiliza para verificar si un ID de reservación es único dentro del vector de reservaciones. Recorre el vector y compara el ID proporcionado con los IDs existentes en el vector. Devuelve true si el ID es único y false si ya existe.

```
/**
 * AgregarLetra: Función para agregar la letra "N" a una cadena
 * @param letra: cadena que almacena la letra "N" (por referencia)
 */
void prjAgregarLetra(string &letra)
{
    bool esLetraValida = false;

    do {
        cout << "Ingrese la letra N: ";
        cin >> letra;
        letra = toupper(letra[0]);

        if (letra == "N") {
            esLetraValida = true;
        } else {
            cout << "Entrada no válida. Ingrese solo la letra N." << endl;
        }
    } while (!esLetraValida);
}
```

La función prjAgregarLetra se utiliza para obtener la letra "N" desde la entrada estándar. Se asegura de que el usuario ingrese solo la letra "N" (mayúscula o minúscula) y almacena la letra en la variable letra utilizando una referencia.

```
/**
 * LeerClientes: Función para leer los clientes desde un archivo
 * @param pathFileName: nombre del archivo
 * @return true si la lectura es exitosa, false en caso contrario
 */
bool prjLeerClientes(string pathFileName)
{
    fstream f;
    string line;
    cout << "Leer archivo: " << pathFileName << endl;
    f.open(pathFileName, ios_base::in);
    if (!f.is_open())
    {
        cout << "Error al abrir el archivo: " << pathFileName << endl;
        return false;
    }
    else
    {
        clientes.clear();
        cout << "Abriendo Archivo: " << pathFileName << endl;
        getline(f, line);
```

```
        while (!f.eof())
        {
            getline(f, line);
            vector<string> vDatos = prjDivirCadena(line, ';');

            Cliente c;
            c.ID = stoi(vDatos.at(0));
            c.Cedula = vDatos.at(1);
            c.Nombre = vDatos.at(2);
            c.Direccion = vDatos.at(3);
            c.Asistencia = stoi(vDatos.at(4));
            clientes.push_back(c);
        }
        return true;
    }
    f.close();
    return false;
}
```

La función prjLeerClientes se utiliza para leer datos de clientes desde un archivo especificado por pathFileName. Abre el archivo, lee cada línea y divide los campos utilizando el carácter ';' como delimitador. Luego, crea instancias de la estructura Cliente y las almacena en el vector clientes. Devuelve true si la lectura es exitosa y false si hay algún problema.

```
/**
 * MostrarClientes: Función para mostrar los clientes en la consola
 */
void prjMostrarClientes()
{
    cout << prjPonerColor(rosa) << endl
         << setw(4) << left << "ID"
         << setw(12) << left << "CEDULA"
         << setw(18) << left << "NOMBRE"
         << setw(12) << left << "DIRECCION"
         << setw(7) << left << "ASISTENCIA"
         << prjPonerColor(azul) << endl;

    for (const auto &cliente : clientes)
    {
        cout << setw(4) << left << cliente.ID
             << setw(12) << left << cliente.Cedula
             << setw(18) << left << cliente.Nombre
             << setw(12) << left << cliente.Direccion
             << setw(10) << left << cliente.Asistencia << endl;
    }
}
```

La función prjMostrarClientes muestra los datos de los clientes almacenados en el vector clientes en la consola. Formatea los datos en columnas con etiquetas y colores específicos.

```
/**
 * AgregarCliente: Función para agregar un cliente al vector de clientes
 */
void prjAgregarCliente()
{
    struct Cliente c;
    fflush(stdin);
    cout << endl;
    int id;
    do {
        prjObtenerNumeroPositivo("Ingrese ID Cliente: ", id);
        if (!prjValidarIDClientes(id, clientes))
        {
            cout << "El ID ya existe. Ingrese un ID único." << endl;
        }
    } while (!prjValidarIDClientes(id, clientes));

    c.ID = id;
    cin.ignore();
    do {
        cout << endl << "Ingrese Cédula del Cliente:  ";
        getline(cin, c.Cedula);
        if (c.Cedula.empty()) {
            cout << "La cédula no puede estar vacía. Ingrese nuevamente." << endl;
        }
    } while (c.Cedula.empty());

    do {
        cout << endl << "Ingrese Nombre del Cliente:  ";
        getline(cin, c.Nombre);
        if (c.Nombre.empty()) {
            cout << "El nombre no puede estar vacío. Ingrese nuevamente." << endl;
        }
    } while (c.Nombre.empty());

    do {
        cout << endl << "Ingrese Dirección del Cliente:  ";
        getline(cin, c.Direccion);
        if (c.Direccion.empty()) {
            cout << "La dirección no puede estar vacía. Ingrese nuevamente." <<
endl;
        }
    } while (c.Direccion.empty());

    cout << endl;
    prjObtenerNumeroPositivo("Ingrese Número de Visitas al Hotel: ",
c.Asistencia);
    cin.ignore();
    clientes.push_back(c);
}
```

La función prjAgregarCliente permite al usuario agregar un nuevo cliente al vector de clientes. Solicita al usuario que ingrese datos como el ID, la cédula, el nombre, la dirección y el número de visitas al hotel, validando que se ingresen correctamente y que el ID sea único en el vector clientes.

```
/**
 * GuardarClientes: Función para guardar los datos de clientes en un archivo
 * @param pathFileName: nombre del archivo
 */
void prjGuardarClientes(string pathFileName)
{
    ofstream f(pathFileName);

    if (!f.is_open()) {
        cout << "Error al abrir el archivo: " << pathFileName << endl;
        return;
    }

    f << "ID;CEDULA;NOMBRE;DIRECCION;ASISTENCIA" << endl;
    int numClientes = clientes.size(); // Obtén el número total de clientes

    for (int i = 0; i < numClientes; i++)
    {
        const auto &cliente = clientes[i];
        f << cliente.ID << ';' << cliente.Cedula << ';' << cliente.Nombre << ';'
        << cliente.Direccion << ';' << cliente.Asistencia;
        if (i < numClientes - 1)
            f << '\n';
    }

    f.close();
}
```

La función prjGuardarClientes se utiliza para guardar los datos de los clientes en un archivo especificado por pathFileName. Abre el archivo y escribe los datos en formato CSV (valores separados por punto y coma) con una línea de encabezado. Luego, recorre el vector clientes y escribe los datos de cada cliente en el archivo.

```
/**
 * ExisteCliente: Función para validar si existe un cliente con un ID dado en las
reservaciones
 * @param ID: ID del cliente
 * @return true si el cliente existe en las reservaciones, false en caso contrario
 */
bool prjExisteCliente(int ID)
{
    for (const auto &reservacion : reservaciones) {
        if (reservacion.ID_Cliente  == ID) {
            return true;
        }
    }
```

```
    return false;
}
```

La función prjExisteCliente se utiliza para verificar si un cliente con un ID específico existe en las reservaciones. Recorre el vector de reservaciones y compara el ID del cliente en cada reservación con el ID proporcionado. Devuelve true si el cliente existe en alguna reservación y false en caso contrario.

```
/**
 * LeerReservaciones: Función para leer las reservaciones desde un archivo
 * @param pathFileName: nombre del archivo
 * @return true si la lectura es exitosa, false en caso contrario
 */
bool prjLeerReservaciones(string pathFileName)
{
    fstream f;
    string line;
    cout << "Leer archivo: " << pathFileName << endl;
    f.open(pathFileName, ios_base::in);
    if (!f.is_open())
    {
        cout << "Error al abrir el archivo: " << pathFileName << endl;
        return false;
    }
    else
    {
        reservaciones.clear();
        cout << "Abriendo Archivo: " << pathFileName << endl;
        getline(f, line);
        while (!f.eof())
        {
            getline(f, line);
            vector<string> vDatos = prjDivirCadena(line, ';');

            Reservacion r;
            r.ID_Cliente = stoi(vDatos.at(0));
            r.Nombre = vDatos.at(1);
            r.Dias = stoi(vDatos.at(2));
            r.TAG = vDatos.at(3);
            r.Check = vDatos.at(4);
            reservaciones.push_back(r);
        }
        return true;
    }
    f.close();
    return false;
}
```

La función prjLeerReservaciones se utiliza para leer datos de reservaciones desde un archivo especificado por pathFileName. Al igual que en la función prjLeerClientes, abre el archivo, lee cada línea, divide los campos

utilizando el carácter ';' como delimitador y crea instancias de la estructura Reservacion. Luego, almacena estas instancias en el vector reservaciones. Devuelve true si la lectura es exitosa y false si hay algún problema.

```
/**
 * MostrarReservaciones: Función para mostrar las reservaciones en la consola
 */
void prjMostrarReservaciones()
{
    cout << prjPonerColor(rosa) << endl
         << setw(11) << left << "ID_CLIENTE"
         << setw(18) << left << "NOMBRE"
         << setw(5) << left << "DIAS"
         << setw(18) << left << "TAG"
         << setw(15) << left << "CHECK"

         << prjPonerColor(azul) << endl;

    for (const auto &reservacion : reservaciones)
    {
        cout << setw(11) << left << reservacion.ID_Cliente
             << setw(18) << left << reservacion.Nombre
             << setw(5) << left << reservacion.Dias
             << setw(18) << left << reservacion.TAG
             << setw(15) << left << reservacion.Check << endl;
    }
}
```

La función prjMostrarReservaciones muestra los datos de las reservaciones almacenadas en el vector reservaciones en la consola. Al igual que en la función prjMostrarClientes, formatea los datos en columnas con etiquetas y colores específicos.

```
/**
 * AgregarReservacion: Función para agregar una reservación al vector de
reservaciones
 */
void prjAgregarReservacion()
{
    struct Reservacion r;
    fflush(stdin);
    cout << endl;
    int id;
    do {
        prjObtenerNumeroPositivo("Ingrese ID Cliente: ", id);
        if (!prjValidarIDReservacion(id, reservaciones))
        {
            cout << "El ID ya existe. Ingrese un ID único." << endl;
        }
    } while (!prjValidarIDReservacion(id, reservaciones));

    r.ID_Cliente = id;
```

```
    cin.ignore();
    do {
        cout << endl << "Ingrese Nombre del Cliente:  ";
        getline(cin, r.Nombre);
        if (r.Nombre.empty()) {
            cout << "El nombre no puede estar vacío. Ingrese nuevamente." << endl;
        }
    } while (r.Nombre.empty());

    cout << endl;
    prjObtenerNumeroPositivo("Ingrese Número de Días: ", r.Dias);
    cin.ignore();
    cout << endl << "Ingrese TAG:  ";
    prjAgregarLetra(r.TAG);
    cout << endl << "Ingrese CHECK:  ";
    prjAgregarLetra(r.Check);
    reservaciones.push_back(r);
}
```

La función prjAgregarReservacion permite al usuario agregar una nueva reservación al vector de reservaciones. Solicita al usuario que ingrese datos como el ID del cliente, el nombre, el número de días, el TAG y el estado de la reservación (CHECK), validando que se ingresen correctamente y que el ID sea único en el vector reservaciones.

```
/**
 * GuardarReservaciones: Función para guardar los datos de reservaciones en un
archivo
 * @param pathFileName: nombre del archivo
 */
void prjGuardarReservaciones(string pathFileName)
{
    ofstream f(pathFileName);

    if (!f.is_open()) {
        cout << "Error al abrir el archivo: " << pathFileName << endl;
        return;
    }

    f << "ID_CLIENTE;NOMBRE;DIAS;TAG;CHECK" << endl;
    int numReservaciones = reservaciones.size(); // Obtén el número total de
reservaciones

    for (int i = 0; i < numReservaciones; i++)
    {
        const auto &reservacion = reservaciones[i];
        f << reservacion.ID_Cliente << ';' << reservacion.Nombre << ';'
          << reservacion.Dias << ';' << reservacion.TAG << ';' <<
reservacion.Check;
        if (i < numReservaciones - 1)
            f << '\n';
    }
```

```
    f.close();
}
```

La función prjGuardarReservaciones se utiliza para guardar los datos de las reservaciones en un archivo especificado por pathFileName. Al igual que en la función prjGuardarClientes, abre el archivo y escribe los datos en formato CSV (valores separados por punto y coma) con una línea de encabezado. Luego, recorre el vector reservaciones y escribe los datos de cada reservación en el archivo.

```
/**
 * LeerTag: Función para leer la línea que comienza con "UID" en el archivo NFC
 * @param ARCHIVOTAG: nombre del archivo
 * @return la línea que comienza con "UID" o una cadena vacía si no se encuentra
 */
string prjLeerTag(string ARCHIVOTAG)
{
    ifstream f(ARCHIVOTAG);
    string line;

    if (!f.is_open())
    {
        cout << "Error al abrir el archivo" << endl;
        return "";
    }

    while (getline(f, line))
    {
        if (line.find("UID") == 0) // Buscar línea que comienza con "UID"
        {
            f.close();
            return line;
        }
    }

    f.close();
    return "";
}
```

La función prjLeerTag se utiliza para leer una línea del archivo especificado por ARCHIVOTAG que comience con la cadena "UID". Abre el archivo, busca la línea que cumple con esta condición y la devuelve. Si no se encuentra una línea que comience con "UID", devuelve una cadena vacía.

```
/**
 * HacerCheckIN: Funcion para cambiar el tag NFC y hacer CHECKIN
 * @param reservaciones: vector de reservaciones
 * @param ID: ID del cliente a cambiar TAG
*/
void prjHacerCheckIN(vector<Reservacion> &reservaciones)
{
    int ID;
```

```
    bool clienteValido = false;

    do {
        prjObtenerNumeroPositivo("Ingrese el ID del cliente: ", ID);

        if (!prjExisteCliente(ID))
            cout << "El cliente con ID " << ID << " no existe." << endl;
         else
            clienteValido = true;

    } while (!clienteValido);

    for (auto &reservacion : reservaciones) {
        if (reservacion.ID_Cliente == ID) {
            reservacion.TAG = prjLeerTag(ARCHIVOTAG);
            reservacion.Check = "Si";
        }
    }
}
```

prjHacerCheckIN: Esta función se encarga de realizar el proceso de check-in para un cliente en el sistema de reservaciones. Primero, solicita al usuario que ingrese el ID del cliente y verifica si dicho cliente existe. Si el cliente es válido, se procede a cambiar su TAG NFC y se registra su entrada en las reservaciones.

```
/**
 *HacerCheckOUT: Funcion para cambiar el tag NFC y hacer CHECKOUT
 * @param reservaciones: vector de reservaciones
 * @param clientes: vector de clientes
 * @param ID: ID del cliente a cambiar TAG
*/
void prjHacerChekOUT(vector<Reservacion> &reservaciones, vector<Cliente>
&clientes)
{
    int ID;
    bool clienteValido = false;

    do {
        prjObtenerNumeroPositivo("Ingrese el ID del cliente: ", ID);

        if (!prjExisteCliente(ID)) {
            cout << "El cliente con ID " << ID << " no existe." << endl;
        } else {
            clienteValido = true;
        }

    } while (!clienteValido);

    for (auto it = reservaciones.begin(); it != reservaciones.end(); ) {
        if (it->ID_Cliente == ID && it->Check == "Si" && it->TAG ==
prjLeerTag(ARCHIVOTAG))
        {
```

```
            it = reservaciones.erase(it); // Elimina el elemento y mueve el
iterador al siguiente

            for (auto &cliente : clientes)
            {
                if (cliente.ID == ID)
                    cliente.Asistencia++;
            }
            prjGuardarClientes(ARCHIVOCLIENTESGENERAL);
        } else {
            cout << "No se pudo realizar el checkout porque no hizo check in o el
TAG no coincide" << endl;
            ++it; // Avanza al siguiente elemento
        }
    }
    prjGuardarReservaciones(ARCHIVORESERVACIONES);
}
```

La función prjHacerChekOUT se encarga del proceso de check-out para un cliente en las reservaciones. Al igual que en el check-in, se solicita al usuario que ingrese el ID del cliente y se verifica su existencia. Luego, se valida si el cliente ha realizado el check-in previamente y si su TAG NFC coincide. En caso de cumplir estas condiciones, se registra su salida y se actualiza la asistencia del cliente.

```
/**
 * MenuPrincipal: Funcion para mostrar el menu principal
 * @param opc: opcion elegida
*/
void prjMenuPrincipal(int &opc)
{
    bool opcValida = false;
    do
    {
        cout << endl << prjPonerColor(rojo) << "\t\t\tMenu Principal" << endl;
        cout << endl << prjPonerColor(azul) << "\t1. Clientes Nuevo" << endl
                                            << "\t2. Cliente Antiguos" << endl
                                            << "\t3. Salir" << endl;
        cout << "\tIngresar Opcion: ";
        try
        {
            cin >> opc;
            cin.clear();
            if (opc>0 && opc<=3)
            {
                opcValida = true;
                break;
            }
            throw invalid_argument("Opcion invalida");
        }
        catch(...)
        {
            cout << "Opcion invalida" << endl;
```

```
            fflush(stdin);
        }
    } while (!opcValida);
}
```

Esta función muestra el menú principal del programa, brindando al usuario varias opciones. El usuario puede optar por gestionar clientes nuevos, clientes antiguos o salir del programa. Además, se encarga de validar la entrada del usuario y garantizar que se elija una opción válida.

```
/**
 * MenuClientesNuevos: Funcion para mostrar el menu de clientes nuevos
 * @param opc: opcion elegida
*/
void prjMenuClientesNuevos(int &opc)
{
    bool opcValida = false;
    do
    {
        cout << endl << prjPonerColor(rojo) << "\t\t\tMenu Clientes Nuevos" <<
endl;
        cout << endl << prjPonerColor(azul) << "\t1. Ingresar Cliente" << endl
                                            << "\t2. Guardar Cliente" << endl
                                            << "\t3. Salir al Menu principal" <<
endl;
        cout << "\tIngresar Opcion: ";
        try
        {
            cin >> opc;
            cin.clear();
            if (opc>0 && opc<=3)
            {
                opcValida = true;
                break;
            }
            throw invalid_argument("Opcion invalida");
        }
        catch(...)
        {
            cout << "Opcion invalida" << endl;
            fflush(stdin);
        }
    } while (!opcValida);
}
```

El propósito de esta función es mostrar un menú específico para la gestión de clientes nuevos en el sistema. Ofrece opciones para ingresar los datos de un nuevo cliente, guardar la información de los clientes ingresados o regresar al menú principal. Al igual que en el menú principal, se asegura de que las opciones ingresadas sean válidas antes de continuar.

```
/**
 * MenuClientesAntiguos: Funcion para mostrar el menu de clientes antiguos
 * @param opc: opcion elegida
*/
void prjMenuClientesAntiguos(int &opc)
{
    bool opcValida = false;
    do
    {
        cout << endl << prjPonerColor(rojo) << "\t\t\tMenu Reservaciones" << endl;
        cout << endl << prjPonerColor(azul) << "\t1. Agregar Reservacion" << endl
                                            << "\t2. CheckIN" << endl
                                            << "\t3. CheckOUT" << endl
                                            << "\t4. Salir al menu principal" <<
endl;
        cout << "\tIngresar Opcion: ";
        try
        {
            cin >> opc;
            cin.clear();
            if (opc>0 && opc<=4)
            {
                opcValida = true;
                break;
            }
            throw invalid_argument("Opcion invalida");
        }
        catch(...)
        {
            cout << "Opcion invalida" << endl;
            fflush(stdin);
        }
    } while (!opcValida);
}
```

prjMenuClientesAntiguos: Esta función se encarga de mostrar el menú para la gestión de clientes antiguos en el sistema de reservaciones. El usuario puede elegir entre varias opciones, como agregar una nueva reservación, realizar check-in, check-out o regresar al menú principal. La función valida la entrada del usuario y garantiza que se seleccione una opción válida.

```
int main ()
{
    cout << LIMPIAR << endl;
    int eleccion = 0;
    bool salir = false, salirClienteNuevo = false, salirClienteAntiguo= false;
    fflush(stdin);
    do
    {
        prjLeerClientes(ARCHIVOCLIENTESGENERAL);
        prjLeerReservaciones(ARCHIVORESERVACIONES);
```

```
        cout << LIMPIAR;
        cout << endl << prjPonerColor(blanco) << "\t\tBienvenido al Sistema de
Reservaciones" << endl;
        cout << "\t   Sistema de reservaciones del hotel \"La Paz\"" << endl;
        prjMostrarClientes();
        prjMostrarReservaciones();
        prjMenuPrincipal(eleccion);
        switch (eleccion)
        {
            case 1:
            {
                int OpcMenuClientesNuevos = 0;
                do
                {
                    cout << LIMPIAR;
                    cout << endl << prjPonerColor(blanco) << "\t\t\tBienvenido al
Sistema de Reservaciones" << endl;
                    cout << "\tSistema de reservaciones del hotel \"La Paz\"" <<
endl;
                    prjMenuClientesNuevos(OpcMenuClientesNuevos);
                    switch (OpcMenuClientesNuevos)
                    {
                        case 1:
                        {
                            prjMostrarClientes();
                            prjAgregarCliente();
                                break;
                        }
                        case 2:
                        {
                            prjGuardarClientes(ARCHIVOCLIENTESGENERAL);
                                break;
                        }
                            case 3: salirClienteNuevo = true;
                                    break;
                    }
                } while (!salirClienteNuevo);
                salirClienteNuevo = false;
                break;
            }
            case 2:
            {
                int OpcReservaciones = 0;
                do
                {
                    cout << LIMPIAR;
                    prjMostrarReservaciones();
                    prjMenuClientesAntiguos(OpcReservaciones);
                    switch (OpcReservaciones)
                    {
                        case 1:
                        {
                            prjMostrarClientes();
                            prjAgregarReservacion();
```

```
                            prjGuardarReservaciones(ARCHIVORESERVACIONES);
                                break;
                        }
                        case 2:
                        {
                            prjHacerCheckIN(reservaciones);
                            prjGuardarReservaciones(ARCHIVORESERVACIONES);
                                break;
                        }
                        case 3:
                        {
                            prjHacerChekOUT(reservaciones, clientes);
                                break;
                        }
                        case 4: salirClienteAntiguo = true;
                                break;
                    }
                } while (!salirClienteAntiguo);
                salirClienteAntiguo = false;
                break;
            }
            case 3:
            {
                salir = true;
                break;
            }
        }
    } while (!salir);
    return 0;
}
```

main(): El flujo principal del programa se encuentra en la función main(). Inicializa las variables necesarias y muestra un mensaje de bienvenida al usuario. Luego, entra en un bucle principal que permite al usuario interactuar con el sistema de reservaciones. Se carga la información de clientes y reservaciones desde archivos, se muestran en pantalla y se presentan las opciones del menú principal. Dependiendo de la opción elegida por el usuario, se llama a funciones específicas para gestionar clientes nuevos o clientes antiguos. El programa continúa ejecutándose hasta que el usuario elige salir.

El programa completo es una aplicación de gestión de reservaciones para un hotel llamado "La Paz". Permite a los usuarios agregar nuevos clientes, registrar reservaciones, realizar check-in y check-out de clientes, y guardar los datos en archivos para su persistencia. La estructura modular de las funciones facilita la interacción con el sistema y garantiza la validación de las operaciones realizadas por el usuario.