Universidad Tecnológica Nacional
Facultad Regional Bahía Blanca

Tecnicatura Universitaria en Programación
Programación I

Proyecto Final

Alumnos:
D'Onofrio, Daniel
Haag Gomez, Gaston

Profesor:
Menvielle, Mateo

Ayudante de Cátedra:
Menna, Ricardo

Resumen:

En el presente informe sobre nuestro proyecto se detallan los pasos realizados durante la programación de un simulador de cajero automático utilizando el lenguaje C++.

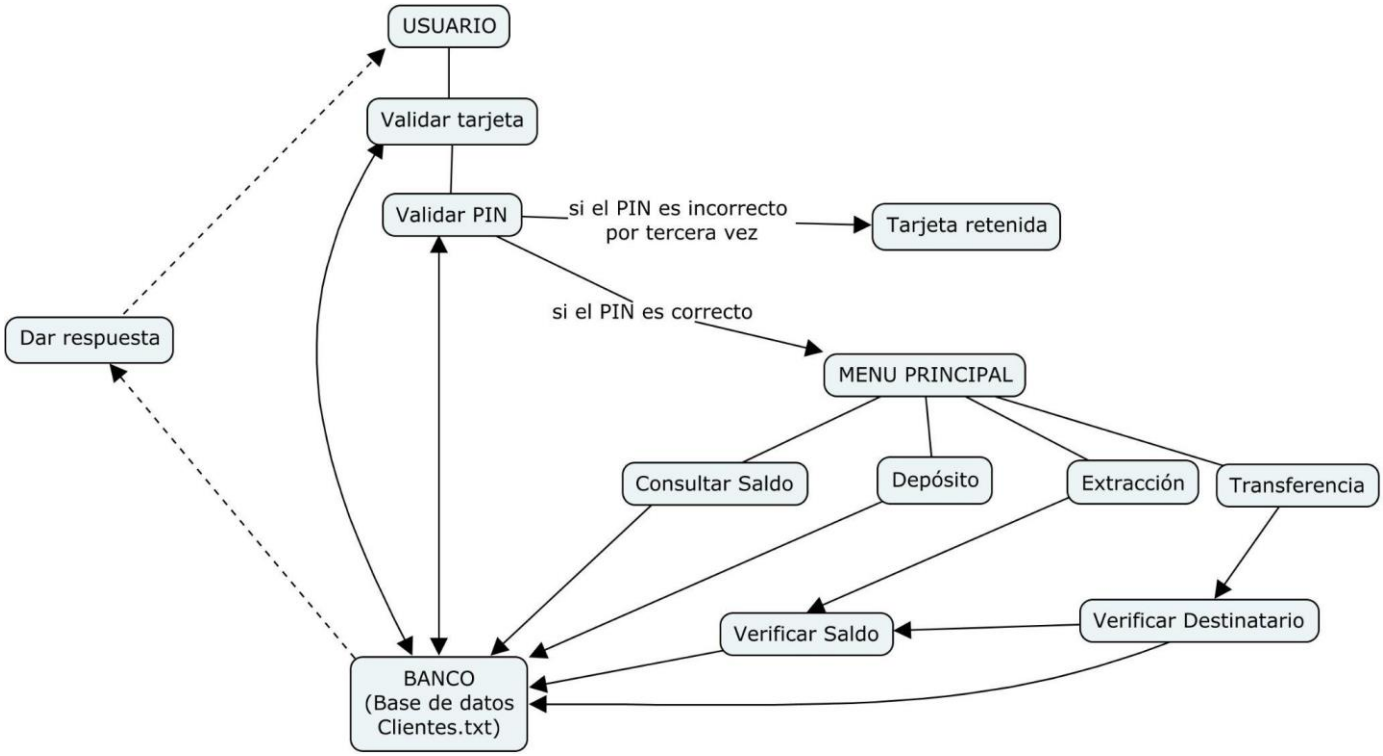
Bahía Blanca | 9 de agosto de 2023

INTRODUCCIÓN:

Se realizó un simulador de cajero automático, para ello se llevó a cabo una investigación sobre el funcionamiento habitual de los cajeros automáticos de nuestro país. Mediante la observación de los mismos, se tuvieron en cuenta las opciones más comunes que se encontraron en ellos, las cuales eran:

- Consultar saldo
- Extraer dinero
- Realizar depósitos de efectivo
- Realizar transferencias a otros usuarios

Con esta información realizamos un diagrama de bloques para diseñar el funcionamiento interno de nuestro programa:



Luego procedemos a detallar cada una de las funciones propuestas para el desarrollo del proyecto y así contemplar las características necesarias para la programación de cada una de ellas.

1. Pide los últimos 4 números de la tarjeta. Verifica si los datos ingresados son correctos (sólo debe aceptar números naturales).
2. Busca el usuario a través del número de la tarjeta en el archivo clientes.txt. Si el número de tarjeta no existe, vuelve a solicitarlo.
3. Pide el pin al usuario. Verifica que los datos ingresados sean correctos (sólo debe aceptar números naturales).
    a. Lee el pin del archivo.

b. Si es incorrecto vuelve a empezar. En caso de que el usuario ingrese un pin incorrecto por tercera vez, su tarjeta será retenida.
c. Si es correcto: Le da la bienvenida al usuario y le muestra el menú principal.

   Menu principal:
   1. Consultar saldo.
   2. Deposito.
   3. Extracción
   4. Transferencia
   5. Cancelar transacción

4. Mostrar saldo: mostrar el saldo de la cuenta actualizado. Preguntar si desea realizar otra transacción (S/N)
5. Depósito de efectivo: suma un importe (que ingresa el usuario) al saldo anterior de la cuenta. Preguntar si desea realizar otra transacción (S/N)
6. Extracción de efectivo: resta un importe (que ingresa el usuario) al saldo anterior de la cuenta. Preguntar si desea realizar otra transacción (S/N)
7. Transferencia: pide al usuario un cbu para realizar la transferencia, pide al usuario el importe a transferir, verifica si el saldo en su cuenta no es menor al saldo que se desea transferir. Busca el destinatario al que se le va a transferir el importe en el archivo clientes.txt:
   a. Si el cliente no existe: mostrar error y preguntar si desea realizar otra transacción (S/N)
   b. Si el cliente existe: mostrar el nombre del destinatario, pedir el importante y solicitar la confirmación de la transferencia (S/N). Luego de realizar la transferencia y preguntar si desea realizar otra transacción (S/N). Mostrar error si el destinatario es el mismo cliente que desea realizar la transferencia.

DESARROLLO:

Todo cajero automático trabaja con una base de datos de los clientes que pertenecen al banco. Para crear nuestro simulador debemos representar de alguna forma dicha base de datos. La misma se encontrará en un archivo llamado clientes.txt que se encuentra junto al código del programa principal. Este archivo contendrá los siguientes datos de cada uno de los clientes del banco: 4 últimos números de la tarjeta del cliente, pin del usuario, nombre y apellido, cuenta bancaria y el saldo de la misma.
Para simplificar la simulación de introducción de tarjeta hemos decidido trabajar solo con los últimos cuatro números de la misma.

```
5000,1234,Pedro Gomez,1234567890,25000
4000,4321,Mario Fernandez,9876543210,30
6000,0123,Alfonzo Perez,0123456789,10000
```

Contenido del archivo clientes.txt

Las librerías que hemos utilizado para el desarrollo del programa son las siguientes:

- ➔ iostream: Fue utilizada para poder trabajar con cin y cout
- ➔ stdlib.h: Fue utilizada para hacer uso de la función exit() la cual nos permitirá salir del programa en cualquier parte del código y la función system() la cual nos permitirá limpiar la pantalla de la consola.
- ➔ conio.h: Fue utilizada para hacer uso de la función getch() la cual nos permite pausar la ejecución hasta que el usuario presione alguna tecla.
- ➔ fstream: Fue utilizada para poder leer y escribir el archivo clientes.txt
- ➔ sstream: Fue utilizada para convertir una variable de string a int
- ➔ cctype: Fue utilizada para hacer uso de la función isdigit() la cual comprueba si lo que ingresa a la misma es un número.
- ➔ algorithm: Fue utilizada para hacer uso de la función all_of()
- ➔ limits: Fue utilizada para hacer uso de numeric_limits

```
#include <iostream>
#include <stdlib.h>
#include <conio.h>
#include <fstream>
#include <sstream>
#include <cctype>
#include <algorithm>
#include <limits>
```

Librerías utilizadas

Para trabajar con los clientes necesitamos crear una estructura. La misma contiene una variable para almacenar el número de la tarjeta del cliente, otra para almacenar el pin, otra para el nombre, otra para el cbu y por último una variable para almacenar el saldo de la cuenta del cliente.

```
struct Cliente {
    int tarjeta;
    int pin;
    string nombre;
    int cbu;
    double saldo;
};
```

Estructura Cliente

Luego definimos las variables a utilizar. Utilizaremos una variable de tipo Cliente para almacenar los datos de la lectura del archivo clientes.txt. Una variable de tipo entero llamada “opción” para almacenar la selección que elija el cliente en el menú principal. Una variable de tipo char llamada “continuar” para

almacenar la decisión del cliente cuando se le consulta si desea realizar otra operación. Una variable de tipo entero llamada “contador”, inicializada en 0, para contar la veces que el cliente introduce un pin incorrecto. Una variable de tipo entero llamada “pinCorrecto” para almacenar de forma temporal el pin que se encuentra en el archivo clientes.txt.

```
    Cliente cliente;
    int opcion;
    char continuar;
    int contador = 0;
    int pinCorrecto;
```

Definición de variables

FUNCIONES:

Función menuPrincipal:

Para mostrar el menú principal creamos una función de tipo void llamada menuPrincipal. La misma muestra en pantalla el menú principal del cajero automático con sus correspondientes opciones: consultar saldo, depósito, extracción, transferencia y cancelar transacción.

```
void menuPrincipal()
{
    cout << "  _____________________________________" << endl
         << " |                                     |" << endl
         << " |         CAJERO AUTOMATICO           |" << endl
         << " |_____________________________________|" << endl
         << " |                                     |" << endl
         << " |  1. Consultar saldo                 |" << endl
         << " |  2. Deposito                        |" << endl
         << " |  3. Extracción                      |" << endl
         << " |  4. Transferencia                   |" << endl
         << " |  5. Cancelar transaccion            |" << endl
         << " |_____________________________________|" << endl
         << endl
         << "Ingrese la opción deseada: ";
}
```

Función buscarUsuario:

Es la función encargada de hacer la búsqueda del usuario en la base de datos de clientes (el archivo clientes.txt). La misma es una función booleana la cual genera una respuesta positiva en caso de que el número de la tarjeta ingresada se encuentre en el archivo de clientes o negativa si el número de la tarjeta no pertenece a ningún cliente. Esto lo realiza a través de tres parámetros, uno pasado por valor y dos por referencia.

tarjetaIngresada (por valor): Es el parámetro encargado de ingresar en la función el número de la tarjeta que introduce el cliente.

pin (por referencia): Este parámetro se encarga de almacenar el pin del usuario en la variable pinCorrecto la cual luego se utilizará en la comprobación del pin.
cliente (por referencia): Este parámetro se encarga de llenar la estructura cliente con los valores contenidos en el archivo clientes.txt.
El recorrido del archivo se realiza a través de un bucle while el cual comprueba cada una de las líneas en búsqueda de la tarjeta ingresada por el cliente. En caso de que encuentre la tarjeta ingresada procederá a llenar la estructura cliente con los datos del mismo a través de los valores pasados por referencia, luego cerrará el archivo y retornará "true". En caso de que no encuentre la tarjeta del cliente la función simplemente cerrará el archivo y retornará false.

```
bool buscarUsuario(int tarjetaIngresada, int& pin, Cliente& cliente)
{
    ifstream archivo;
    archivo.open("clientes.txt", ios::in);

    if (archivo.fail())
    {
        cout << "No se pudo abrir el archivo de clientes." << endl;
        exit(1);
    }

    string linea;
    char delimitador = ',';
    // Leemos todas las líneas
    while (getline(archivo, linea))
    {
        stringstream stream(linea); // Convertir la cadena a un stream
        string tarjetaCorrecta, pinCorrecto, cbu, nombre, saldo;

        getline(stream, tarjetaCorrecta, delimitador);
        int tarjeta = stoi(tarjetaCorrecta);

        if (tarjeta == tarjetaIngresada)
        {

            cliente.tarjeta = tarjeta;

            getline(stream, pinCorrecto, delimitador);
            istringstream(pinCorrecto) >> pin;

            getline(stream, cliente.nombre, delimitador);

            getline(stream, cbu, delimitador);
            istringstream(cbu) >> cliente.cbu;
```

```
            getline(stream, saldo);
            istringstream(saldo) >> cliente.saldo;

            archivo.close();
            return true;
        }
    }
    archivo.close();
    return false;
}
```

Función verificar PIN:

La función verificar PIN se encarga de comparar el PIN ingresado por el cliente con el PIN correcto almacenado en el archivo clientes .txt. Esta función utiliza dos parámetros pasados por valor: pin, donde se encontrará el pin ingresado por el cliente, y pinArchivo donde se encontrará el pin del cliente almacenado en clientes.txt.

```
bool verificarPin(int pin, int pinArchivo)
{
    return (pin == pinArchivo);
}
```

Función mostrarSaldo:

Esta función es la encargada de mostrar el saldo del cliente en pantalla. Muestra el saldo que se encuentra almacenado en la estructura cliente, la cual había sido previamente completada con los valores del archivo al ejecutar la función buscarUsuario.

```
void mostrarSaldo(Cliente cliente)
{
    cout << "Su saldo actual es: " << cliente.saldo << endl;
}
```

Función realizarDepósito:

Esta función se encarga de realizar un depósito en la cuenta del cliente, definimos una variable de tipo entero para almacenar el monto del depósito y creamos un bucle do while para que pida al usuario ingresar el monto a depositar hasta que el monto ingresado sea válido en la función booleana comprobarDatoValido. Luego, suma el monto del depósito al saldo del cliente, actualiza el archivo cliente y por último muestra que el depósito se ha realizado correctamente junto con el saldo actual de la cuenta.

```
void realizarDeposito(Cliente& cliente) {
    int deposito;
    do
    {
        cout << "Ingrese el importe a depositar: ";
```

```
        cin >> deposito;
    } while (!comprobarDatoValido(deposito));
    cliente.saldo += deposito;
    actualizarArchivo(cliente);
    cout << "Deposito realizado correctamente. Saldo actual: " << cliente.saldo
<< endl;
}
```

Función Realizar Extracción:

Esta función es similar a la anterior y se encarga de realizar una extracción en la cuenta del cliente. Definimos una variable de tipo entero para almacenar el monto de la extracción y creamos un bucle do while para que pida al usuario ingresar el monto a depositar hasta que el monto ingresado sea válido en la función booleana comprobarDatoValido. Luego, a través de un condicional se verifica el monto y en caso de que el mismo sea mayor al saldo del cliente muestra el error "Saldo insuficiente para realizar la extracción". En caso contrario realizará la extracción y procederá a actualizar el archivo cliente.txt. Por último imprime en pantalla que la extracción se ha realizado correctamente junto con el saldo actual de la cuenta.

```
void realizarExtraccion(Cliente& cliente) {
    int extraccion;

    do
    {
        cout << "Ingrese el importe que desea extraer: ";
        cin >> extracción;
    } while (!comprobarDatoValido(extracción));

    if (extracción > cliente.saldo) {
        cout << "Error: Saldo insuficiente para realizar la extracción." << endl;
    }
    else {
        cliente.saldo -= extracción;
        actualizarArchivo(cliente);
        cout << "Extracción realizada correctamente. Saldo actual: " <<
cliente.saldo << endl;
    }
}
```

Función Realizar Transferencia:

Esta función es la encargada de realizar las transferencias entre diferentes clientes. Para ello se comienza solicitando al cliente el CBU del destinatario. El mismo se almacena en una variable de tipo entero y se comprueba a través de la función comprobarDatoValido, la cual se encargará de que el cliente no ingrese caracteres incorrectos. En caso de que el CBU sea el mismo que posee el cliente se mostrará un error en

pantalla diciendo al usuario que no se puede transferir a sí mismo y se saldrá de la función a través de return.
Luego se solicitará al usuario que ingrese el importe a transferir. El mismo también se almacena en una variable de tipo double y se comprueba a través de la función comprobarDatoValido. En caso de que el importe ingresado sea mayor al saldo que posee el cliente se mostrará un error diciendo que su saldo es insuficiente de lo contrario se procederá a realizar la transferencia.
Para realizar la transferencia se comienza creando una nueva estructura Cliente denominada "destinatario". A través de la función buscarDestinatarioPorCBU se comprobará que el CBU ingresado pertenezca a un destinatario válido y se almacenarán los datos del mismo en la estructura "destinatario". En caso de que el destinatario no exista se mostrará un error al usuario y en caso de que sí exista se mostrará el nombre del destinatario y se le pedirá al usuario que confirme la transferencia. Una vez que el usuario confirma la transferencia al saldo almacenado en la estructura cliente se le resta el importe de la transferencia y al saldo almacenado la estructura "destinatario" se le suma el importe de la transferencia. Finalmente llamando a la función actualizarArchivo se procede a actualizar los datos en el archivo clientes.txt tanto del cliente que realizó la transferencia como también del destinatario de la misma.

```
void realizarTransferencia(Cliente& cliente) {
    int cbu;
    double transferencia;
    do
    {
        cout << "Ingrese el CBU del destinatario: ";
        cin >> cbu;
    } while (!comprobarDatoValido(cbu));

    if (cbu == cliente.cbu) {
        cout << "Error: No se puede transferir al mismo CBU." << endl;
        return; // Salir de la función sin continuar con la transferencia
    }

    do
    {
        cout << "Ingrese el importe a transferir: ";
        cin >> transferencia;
    } while (!comprobarDatoValido(transferencia));

    if (transferencia > cliente.saldo) {
        cout << "Error: Saldo insuficiente para la transferencia." << endl;
    }
    else {
        Cliente destinatario;
        if (buscarDestinatarioPorCBU(cbu, destinatario)) {
            cout << "Destinatario: " << destinatario.nombre << endl;
```

```
            char confirmacion;
            cout << "Confirmar transferencia (S/N): ";
            cin >> confirmación;
            if (confirmacion == 'S' || confirmacion == 's') {
                cliente.saldo -= transferencia;
                destinatario.saldo += transferencia;
                actualizarArchivo(cliente);
                actualizarArchivo(destinatario);
                cout << Transferencia realizada correctamente." << endl;
            }
            else {
                cout << "Transferencia cancelada." << endl;
            }
        }
        else {
            cout << "Error: El destinatario no existe." << endl;
        }
    }
}
```

Función Actualizar Archivo:

Esta función es la encargada de actualizar los datos de los clientes en el archivo clientes.txt. Para su funcionamiento se utiliza la librería fstream.

Al inicio se declara “ifstream archivoIn”, quien se encargará de leer el archivo existente llamado "clientes.txt". Por otra parte declaramos “ofstream archivoOut” para poder crear un archivo temporal “temp.txt” con los datos a reemplazar en el archivo original.

Luego se abre el archivo existente en modo lectura y con un condicional if se analiza si la apertura falla. En caso de que falle se mostrará en pantalla un error. De la misma forma se procede con el archivo temporal.

Creamos una variable de tipo string para almacenar las líneas del archivo original y otro para almacenar el número de la tarjeta del cliente y compararlo con el número almacenado en la estructura cliente.

Nuestro archivo clientes.txt se encuentra fragmentado a través de un delimitador el cual en nuestro caso es la coma. Para indicarle a la función que lea y escriba los datos de forma fragmentada a través de dicho delimitador utilizamos stringstream stream() sobre la variable línea que creamos anteriormente. Esto dividirá la línea en varios campos. Luego se utiliza la función getline(stream, dato, ',') para almacenar dichos campos en la variable dato.

Procedemos a convertir el número de la tarjeta del cliente a tipo entero y realizamos la comparación entre el número que se encuentra en la estructura y el número que se encuentra en la línea que se está leyendo actualmente. En caso de que el número de tarjeta coincida se procede a actualizar los datos de la línea con los que se encuentran almacenados en la estructura que ingresa a través del parámetro. En caso de que el número de la tarjeta no coincida simplemente se escribirá lo que se almaceno en la variable línea sin realizar ninguna modificación.

Una vez que ya tenemos creado nuestro archivo temp.txt con los datos actualizados simplemente procedemos a realizar un intercambio de archivos. Para ello borraremos el archivo clientes.txt con la función remove() y utilizaremos la función rename() para cambiar el nombre del archivo temp.txt por clientes.txt

```
void actualizarArchivo(Cliente& cliente) {
    ifstream archivoIn;
    ofstream archivoOut;

    archivoIn.open("clientes.txt", ios::in);
    if (archivoIn.fail()) {
        cout << "No se pudo abrir el archivo de clientes." << endl;
        exit(1);
    }

    archivoOut.open("temp.txt", ios::out);
    if (archivoOut.fail()) {
        cout << "Error al crear archivo temporal." << endl;
        archivoIn.close();
        exit(1);
    }

    string linea;
    while (getline(archivoIn, linea)) {
        stringstream stream(linea);
        string dato;

        getline(stream, dato, ',');
        int tarjeta = stoi(dato);

        if (tarjeta == cliente.tarjeta) {
            archivoOut << tarjeta << "," << cliente.pin << "," << cliente.nombre
<< "," << cliente.cbu << "," << cliente.saldo << endl;
        }
        else {
            archivoOut << linea << endl;
        }
    }

    archivoIn.close();
    archivoOut.close();

    remove("clientes.txt");
    rename("temp.txt", "clientes.txt");
}
```

Buscar Destinatario por CBU:

Esta función se creó para poder buscar al usuario a través de su CBU en vez de a través de su tarjeta como realiza la función buscarUsuario. El funcionamiento de la misma es igual al de buscarUsuario solo que al final del ciclo while se agrega un condicional if que compara el valor de destinatario.cbu con el del cbuIngresado y si son iguales nos devolverá true.

```
bool buscarDestinatarioPorCBU(int cbuIngresado, Cliente& destinatario)
{
    ifstream archivo;
    archivo.open("clientes.txt", ios::in);

    if (archivo.fail()) {
        cout << "No se pudo abrir el archivo de clientes." << endl;
        exit(1);
    }

    string linea;
    char delimitador = ',';
    // Leemos todas las líneas
    while (getline(archivo, linea))
    {
        stringstream stream(linea); // Convertir la cadena a un stream
        string tarjetaCorrecta, pinCorrecto, cbu, nombre, saldo;

        getline(stream, tarjetaCorrecta, delimitador);
        int tarjeta = stoi(tarjetaCorrecta);

        destinatario.tarjeta = tarjeta;

        getline(stream, pinCorrecto, delimitador);
        istringstream(pinCorrecto) >> destinatario.pin;

        getline(stream, destinatario.nombre, delimitador);

        getline(stream, cbu, delimitador);
        istringstream(cbu) >> destinatario.cbu;

        getline(stream, saldo);
        istringstream(saldo) >> destinatario.saldo;

        if (destinatario.cbu == cbuIngresado) {
            archivo.close();
            return true;
        }
    }
```

```
    archivo.close();
    return false;
}
```

Función comprobarDatoValido:

En esta función booleana iniciamos asumiendo que el número no es válido, para luego abrir un condicional if que verifique el fallo en la entrada en la cual, si devuelve verdadero significa que hubo un fallo en la lectura de entrada (por ejemplo, cuando se ingresa una letra en lugar de un número) y muestra un mensaje de error y se realiza un pausa en la ejecución con _getch() espera a que el usuario presione una tecla antes de continuar. También realizó un cin.clear() se utiliza para restablecer el estado del flujo de entrada y un cin.ignore(numeric_limits<streamsize>::max(), '\n') se usa para ignorar caracteres restantes en la línea, luego si el número es menor a cero muestra dato ingresado no es válido intente nuevamente sino el número se considera válido (devuelve true), y por último retorna al numeroValido.

```
bool comprobarDatoValido(int numero) {
    bool numeroValido = false;
    if (cin.fail()) {
        cout << "Error: Solo se aceptan valores numéricos. Por favor intente
nuevamente..." << endl;
        _getch();
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    } else if (numero < 0) {
        cout << "El dato ingresado no es válido. Por favor intente nuevamente..."
<< endl;
        _getch();
    }
    else {
        numeroValido = true;
    }
    return numeroValido;
}
```

PROGRAMA PRINCIPAL:

Bucle de verificación de tarjeta:

El programa principal inicia con el bucle de comprobación de tarjeta. El mismo es el encargado de solicitar al usuario que ingrese los cuatro últimos dígitos de su tarjeta. Luego del ingreso un condicional comprobará si los caracteres ingresados son correctos utilizando la función all_of() de la biblioteca <algorithm> la cual se encarga de verificar si los datos ingresados en la variable tarjetaIngresada son dígitos o no. En caso de que no lo sean se le mostrará al usuario un error y se le solicitará que presione una tecla para intentar nuevamente. En caso de que los datos sean correctos se procederá a ejecutar la

función buscarUsuario para comprobar el número de la tarjeta ingresada. Todas estas acciones se repiten en un ciclo do while el cual seguirá actuando hasta que buscarUsuario devuelva verdadero.

```
    do
    {
        cout << "Ingrese los ultimos 4 digitos de su tarjeta: ";
        string tarjetaIngresada;
        cin >> tarjetaIngresada;

        if (!all_of(tarjetaIngresada.begin(), tarjetaIngresada.end(), ::isdigit))
        {
            cout << endl
                << "Error: Los datos ingresados no son válidos. Solo se aceptan
números enteros positivos."
                << endl
                << endl
                << "Presione cualquier tecla para intentar nuevamente...";
            _getch();
            system("cls");
            continue;
        }

        cliente.tarjeta = stoi(tarjetaIngresada);

        if (!buscarUsuario(cliente.tarjeta, pinCorrecto, cliente))
        {
            cout << endl
                << "Numero de tarjeta no existe. Vuelva a intentar."
                << endl
                << endl
                << "Presione cualquier tecla para intentar nuevamente...";
            _getch();
            system("cls");
            continue;
        }
        system("cls");
    } while (!buscarUsuario(cliente.tarjeta, pinCorrecto, cliente));
```

Bucle de verificación de PIN:

Una vez verificada la tarjeta del usuario se procede a la solicitud del pin.

Este ciclo funciona de la misma manera que el ciclo anterior. Solicitará el pin al cliente, comprobará si los caracteres ingresados son correctos utilizando la funcion all_of() y se reperirá en el caso de que los datos no sean válidos. Pero a diferencia del ciclo anterior, una vez que el usuario ingrese datos válidos (número enteros positivos) se llamará a la función verificarPIN y en caso de que el PIN no sea correcto se modificará una variable númerica que hemos creado llamada contador. La misma será la encargada de

contar cuántas veces el usuario ingresa el pin de forma incorrecta. En caso de que el usuario ingrese 3 veces el pin de forma incorrecta el programa mostrará un mensaje diciendo “Su tarjeta ha sido retenida” y saldrá del programa a través de la función exit(1). La comprobación de la cantidad de veces que el usuario ingresa un pin invalido se realiza a través de un condicional que compara el estado actual de la variable contador y verifica que no sea mayor o igual a 3.
Por otro lado se agregó otra función como: system("cls") que se encarga de limpiar la pantalla para que sea una interfaz más limpia antes de solicitar nuevamente el PIN.

```
    do
    {
        cout << "Ingrese su PIN: ";
        string pinIngresado;
        cin >> pinIngresado;

        if (!all_of(pinIngresado.begin(), pinIngresado.end(), ::isdigit))
        {
            cout << endl
                << "Error: Solo se aceptan valores numéricos."
                << endl
                << endl
                << "Presione cualquier tecla para intentar nuevamente...";
            _getch();
            system("cls");
            continue;
        }

        cliente.pin = stoi(pinIngresado);

        if (!verificarPin(cliente.pin, pinCorrecto))
        {
            contador++;

            if (contador >= 3)
            {
                cout << endl
                    << "Su tarjeta ha sido retenida."
                    << endl;
                exit(1);
            }
            cout << endl
                << "PIN incorrecto."
                << endl
                << endl
                << "Presione cualquier tecla para intentar nuevamente...";
            _getch();
            system("cls");
```

```
            continue;
        }
        system("cls");
    } while (!verificarPin(cliente.pin, pinCorrecto));
```

Bucle para mostrar Menú Principal:

Una vez que se ha ingresado un pin correcto se ingresa al menú principal. El mismo saluda al usuario con su nombre el cual es tomado de la estructura cliente (la cual se completó durante la ejecución de la función buscarUsuario). Posteriormente se llama a la función menuPrincipal() para que muestre el menú y se registra la opción que ingrese el usuario en la variable opción. Los datos ingresados en la misma también son verificados como en los bucles anteriores. En este caso se crea un condicional que verifica a través de cin.fail que no se ingresen ningún otro tipo de datos que no sean numéricos. A su vez también se comprueba que los números ingresados estén en un rango de entre 1 y 5 (las cuales son las únicas opciones posibles que muestra el menú principal.

```
do
        {
            system("cls");
            cout << "Bienvenido, " << cliente.nombre << "!" << endl;
            menuPrincipal();
            cin >> opcion;

            if (cin.fail()) {
                cout << endl
                    << "Error: Solo se aceptan valores numéricos."
                    << endl
                    << endl
                    << "Presione cualquier tecla para intentar nuevamente...";
                _getch();
                system("cls");
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                opcion = -1; // Asignamos -1 para que repita el bucle
                continue;
            } else if (opcion < 1 || opcion > 5)
            {
                cout << endl
                    << "Error: Solo se aceptan números entre 1 y 5."
                    << endl
                    << endl
                    << "Presione cualquier tecla para intentar nuevamente...";
                _getch();
                system("cls");
            }
        } while (opcion < 1 || opcion > 5);
```

Switch del Menu Principal:

En el menú principal usamos switch (opción): Esto inicia una estructura de control switch basada en el valor de la variable opción. La función switch se utiliza para tomar decisiones en función de diferentes casos como el nuestro :

case 1: Este es un caso del switch que se ejecutará si la opción es igual a 1. Lo que haces es llamar a la función mostrarSaldo(cliente) para mostrar el saldo del cliente.

case 2: Este es un caso del switch que se ejecutará si la opción es igual a 2. Esta llama a la función realizarDeposito(cliente) para realizar un depósito en la cuenta del cliente.

case 3: Este es un caso del switch que se ejecutará si la opción es igual a 3. Acá llama a la función realizarExtraccion(cliente) para realizar una extracción de la cuenta del cliente.

case 4: Este es un caso del switch que se ejecutará si la opción es igual a 4.Llama a la función realizarTransferencia(cliente) para realizar una transferencia desde la cuenta del cliente.

case 5: Este es un caso del switch que se ejecutará si la opción es igual a 5. Muestra un mensaje de despedida y luego llama a exit(1) para terminar el programa.

default: Esto se ejecutará si la opción no coincide con ninguno de los casos anteriores. Muestra un mensaje de error y por último espera a que el usuario presione una tecla con _getch() antes de continuar.

```
switch (opcion)
{
case 1:
    mostrarSaldo(cliente);

    break;
case 2:
    realizarDeposito(cliente);

    break;
case 3:
    realizarExtraccion(cliente);

    break;
case 4:
    realizarTransferencia(cliente);

    break;
case 5:
    cout << "Muchas gracias por utilizar nuestros servicios…" << endl;
    exit(1);
    break;
default:
    cout << endl
        << "Error: la opción ingresada no es válida."
```

```
                << endl
                << endl
                << "Presione una tecla para intentar nuevamente...";
            _getch();
            break;
        }
```

Bucle para consultar si se desea realizar otra transacción:

Al finalizar cada operación se entra a este ciclo el cual preguntará si se desea realizar otra transacción. El ciclo se repetirá mientras que el usuario no ingrese N o S. Al igual que en los demás ciclos, también se verifica que el usuario ingrese valores correctos en su respuesta.

```
do
        {
            cout << endl << "Desea realizar otra transacción? (S/N): ";
            cin >> continuar;
            continuar = toupper(continuar); // Convertir a mayúscula (

            if (continuar != 'N' && continuar != 'S') {
                cout << "Error: la opción ingresada no es válida." << endl;
                cin.clear(); // Limpiar el estado de error del cin
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignorar
el resto de la línea inválida
            }
        } while (continuar != 'N' && continuar != 'S');
```

Bucle del programa principal:

Todos los bucles anteriores conforman la ejecución del programa principal el cual se ejecutará de forma continua siempre que la variable continuar sea diferente a “N”. Esto asegura tanto una ejecución continua del programa como la salida del programa cuando el usuario ingresa “N” en la pregunta que se realiza en el ciclo anterior.

```
    do
        {
          // Bucle para mostrar menú principal
          // Switch del menú principal
          // Bucle para consultar por otra transacción
        } while (continuar != 'N');
```

CONCLUSIÓN:

En el presente proyecto se elaboró un simulador de cajero automático, para ello fue necesario investigar varias cuestiones que no fueron vistas durante el desarrollo de la materia. El manejo de archivos fue el eje principal del proyecto, como también investigar sobre la inclusión de nuevas funciones utilizando librerías no vistas en la materia.

El desarrollo de la función de transferencia fue el desafío más importante del proyecto, ya que la misma debía actualizar de forma simultánea tanto el saldo del cliente original como el saldo del destinatario en el archivo clientes.txt.

Para concluir valoramos la realización del presente proyecto ya que a través del mismo hemos adquirido importantes conocimientos sobre el manejo de archivos en el lenguaje C++.