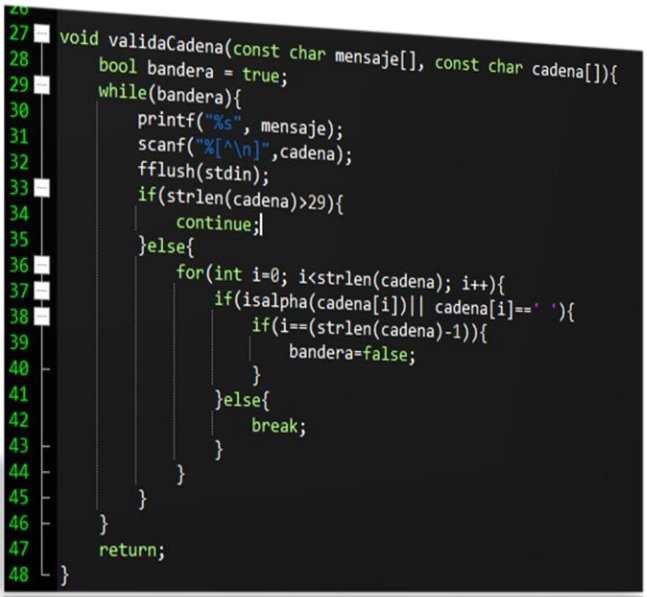

Manual del programador

Autores:
Montejano Briano Uriel – 177771
Bruno Axel Puente Luna – 177876

Programación II
Prof. Guadalupe Ledesma Ramos
Proyecto Final
23/11/2022

Contenido

Tabla de ilustraciones

INTRODUCCIÓN

En este manual se describirá el código cuya función es llevar los registros de un banco. Mediante el uso de 2 tipos de usuarios, el administrador, encargado de registrar los usuarios y modificar los datos de estos mismos; y el ejecutivo, este llevara los registros de los clientes y los contratos disponibles.

MARCO TEÓRICO

Para el desarrollo de este programa, se uso el lenguaje “C”, y utilizamos las listas, listas de listas y lectura de archivos para llevar a cabo el registro de los clientes y usuarios.
Las listas son el mejor método para este tipo de programas, ya que permite reservar el espacio suficiente en la memoria para poder registrar los datos de cada estructura, añadir registros sin la necesidad de cambiar parámetros en arreglos u otros métodos de guardar la información y su facilidad para recorrerse en las listas y listas de listas (facilita la consulta de los datos almacenados).
Las listas son manejadas con la reserva de memoria, a esto se le llama “memoria dinámica”, ya que esta en constante cambio dependiendo de los datos que se almacenarán (los nodos que compondrán a la lista), no son estáticos como lo son los arreglos.

ARCHIVOS EN EL PROYECTO

Para la realización de este proyecto. Se utilizaron las siguientes hojas de trabajo:

- main.cpp
- archivos.h
- archivosCli.h
- decoracion.h
- librería.h
- validacion.h

Cada una de estas se explicarán más adelante.

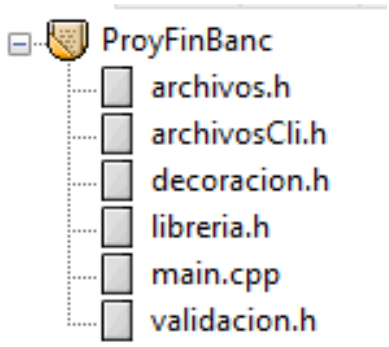

Ilustración 1 Hojas del proyecto

libreria.h

```
1   #include <iostream>
2   #include <stdlib.h>
3   #include <iomanip>
4   #include <locale.h>
5   #include <windows.h>
6   #include <string>
7   #include <fstream>
8   #include <stdio.h>
9   #include <time.h>
10  #include <conio.h>
11  #include <ctype.h>
12  #include <string.h>
13  #include <istream>
14  #include <sstream>
```

Ilustración 2 Bibliotecas

Bibliotecas:

Se declararon las siguientes bibliotecas, las cuales son necesarias para el correcto funcionamiento del programa y utilizar funciones de C++ que facilitaran la programación en gran parte del código.

```
16  using namespace std;
17
18  string estado="Activo",Ahorro="Ahorro",Credito="Credito",Debito="Debito",Creditohipotecario="Cre_hipo";
19  string ahorradOR="ahorrador",Ahorrador="Ahorrador",deudOR="deudor",DEudor="Deudor",Activo="Activo",Inactivo="Inactivo";
20  string En_Deuda="En deuda",Cancelado="Cancelado",Sin_Deuda="Sin Deuda",Administrador="Administrador",Ejecutivo="Ejecutivo";
21  int long long cardnumber;
22
23  bool check;
```

Ilustración 3 Variables globales

Declaración de variables:

Declaración de using namespace std para su uso en C++. Así mismo se declararon variables globales para no tener que definirlas en distintas funciones.
Las variables tipo string serán concatenadas en las funciones, esto para evitar escribir manualmente ciertos valores que serán estándar a lo largo del registro de clientes y usuarios.
La variable tipo entero long long será definida para reservar el espacio suficiente para los caracteres de una tarjeta (ya sea débito, crédito, etc.).
La variable bool, será de utilidad para confirmar si se cumplieron ciertos procesos en las diferentes funciones que se presentan en el proyecto.

```
22  struct Usuario{
23      int idUsuario;
24      string nombre;
25      string apellido;
26      int edad;
27      string telefono;
28      string correo;
29      string dir;
30      string username;
31      string pass;
32      string estado;
33      string perfil;
34
35      Usuario *sig;
36  }*primero, *ultimo;
37
38  struct Contrato{
39      string estadoContra;
40      string tipClien;
41      string tipotarcli;
42      string hipotecaref;
43      string fechaA;
44      float CanAhorro;
45      int interes;
46      float prestamo;
47      float anio;
48  };
```

Ilustración 4 Estructura usuario y contrato

```
50  struct Cliente{
51      int idCliente;
52      string nombrecli;
53      string apellidocli;
54      string telefonocli;
55      string refecli;
56      string tel_refecli;
57      int long long tarjetacli;
58      Contrato deal;
59
60      Cliente *next;
61  }*primerocli, *ultimocli;
62
63  struct Listas{
64      int idListas;
65      Usuario *lista;
66      Listas *sig;
67  }*primLista, *ultLista;
68
69  struct ListasClientes{
70      int idListasCli;
71      Cliente *listacli;
72      ListasClientes *next;
73  }*primListacli, *ultListacli;
```

Ilustración 5 Estructura cliente, listas y listas de clientes

Definición de estructuras:

Estas estructuras son nuestras listas y listas de listas:

Listas:

- Usuario
- Cliente

Listas de listas:

- Listas
- ListasClientes

La estructura contrato solo servirá para guardar los datos que se usaran al registrar los contratos de los clientes, para esta estructura no será necesaria la reserva de memoria.

Las Listas, además de contener los datos que tendrán cada uno de los registros, tienen un espacio para guardar la dirección de memoria, esta dirección de memoria es un apuntador del tipo de la estructura.
Estos apuntadores guardaran la dirección de memoria del siguiente nodo (uso de listas). Además de la dirección de memoria que contiene la estructura, cuenta con 2 apuntadores extra (primero y ultimo), los cuales servirán como auxiliares para guardar las posiciones del primer y último nodo de su respectiva lista (los apuntadores guardan direcciones de memoria).

En las listas de listas incluyen un entero (el id de las listas, que será una número para identificarlas sin la necesidad de las direcciones de memoria), un apuntador del tipo de las listas que se almacenaran (en el caso de la lista de listas “Listas”, guardara los datos de los usuarios), y un apuntador del tipo de la estructura de lista de listas (en el caso de la lista de listas “Listas”, guarda su propia dirección de memoria), estas funcionan de la misma forma que en una lista normal. Al igual que en las listas, también se declaran 2 apuntadores extra que ya no pertenecen a los datos de la estructura, estos guardar la posición del primer y último nodo de las listas de listas (los apuntadores guardan direcciones de memoria).

```
80  void gotoxy(int x,int y)
81   {
82   HANDLE hcon;
83   hcon = GetStdHandle(STD_OUTPUT_HANDLE);
84   COORD dwPos;
85   dwPos.X = x;
86   dwPos.Y= y;
87   SetConsoleCursorPosition(hcon,dwPos);
88   }
```

Ilustración 6 Función gotxy

Función gotoxy:

Se declara la función tipo void “gotoxy”, la cual servirá para usar los comandos de gotoxy (go to x y), cuya función es ubicarse en puntos específicos de la pantalla y cambio de colores de texto, estas funciones tendrán uso decorativo en el programa.

```
91  #include "validacion.h"
92  #include "decoracion.h"
93  #include "archivosCli.h"
94  #include "archivos.h"
```

Ilustración 7 Declaración hojas de trabajo

Declaración hojas de trabajo:

Los include mostrados anteriormente, son declaraciones de otras hojas de trabajo que se encuentran integradas en el proyecto, esto es necesario para que todas las funciones, variables y estructuras declararas en la hoja “librería.h” y ahorrar la declaración en cada una de las hojas con las que se trabajaran en el proyecto.

main.cpp

```
8   #include "libreria.h"
9
10  int login(Usuario *primero,string usuario,string contrasena){
11      struct Usuario *login=primero;
12      int opc;
13      while(login!=NULL){
14          if((usuario==login->username) and (contrasena==login->pass) and (login->perfil==Administrador)){
15              fflush(stdin);
16              system("cls");
17              menuadmin(login->nombre);
18              break;
19          }
20          if((usuario==login->username) and (contrasena==login->pass) and (login->perfil==Ejecutivo)){
21              fflush(stdin);
22              system("cls");
23              menuejec(login->nombre);
24              break;
25          }
26          if(login==NULL){
27              break;
28          }
29          login=login->sig;
30      }
31  }
```

Ilustración 8 Funcion login

En la hoja de trabajo main, se encontrará la estructura principal del programa, es desde aquí donde se llamarán la mayoría de las funciones. Aquí se declara la librería, que contiene las funciones de las demás hojas de trabajo del proyecto.

Función login:

En la función int login, se hará una validación de datos, la cual recibe los parámetros de la dirección de memoria del primer nodo de la lista “Usuarios”, y los string usuario y contraseña.

Definirá un nuevo apuntador que servirá como auxiliar para recorrerse en la lista, guardando la dirección de memoria del primer nodo.

El ciclo while recorrerá el apuntador login mientras sea diferente de NULL, mientras se recorre comparará que los datos registrados estén en el archivo, y dependiendo de si es administrador o ejecutivo mandará a llamar al menú correspondiente; “fflush(stdin)” realiza una limpieza de teclado para evitar errores al momento de cambiar al menú correspondiente.

Ambos menú (menuadmin y menuejec), llevaran el nombre del usuario que inicien sesión, esto para dar un mensaje personalizado en el menú que se vaya a presentar en pantalla.

La línea de código 29 (login=login->sig) es la encargada de recorrer las direcciones de memoria, ya que la dirección de memoria actual tomará la dirección siguiente del actual, esta es la importancia de incluir el apuntador del mismo tipo de la estructura con la que se está trabajando.

```
33  int main(){
34      PrincipioMenu();
35
36      primLista=NULL;
37      ultLista=NULL;
38
39      primListacli=NULL;
40      ultListacli=NULL;
41
42      lecturaLista();
43      lecturaListaCli();
44
45      int cont=0,hold=1,opc;
46      while(hold==1){
47          system("cls");
48          decoprint();
49          cuadrologin();
50          string usuario,contrasena;
51          gotoxy(50,12);
52          cout<<"Usuario: ";
53          cin>>usuario;
54          gotoxy(50,14);
55          cout<<"Contraseña: ";
56          cin>>contrasena;
57
58          struct Listas *aux=primLista;
59          while(aux!=NULL){
60              check=login(aux->lista,usuario,contrasena);
61              aux=aux->sig;
62          }
63          if(check==false){
64              cout<<"Ingrese un usuario valido"<<endl;
65          }
66          if(check==true){
67              cout<<"Hasta luego"<<endl;
68          }
69          aux=primLista;
70          system("cls");
71          decoprint();
72          cuadrologin();
```

Ilustración 9 Función main

```
72          cuadrologin();
73          gotoxy(50,12);
74          cout<<"1.-Ingresar a una cuenta"; gotoxy(50,13); cout<<"2.-Salir"; gotoxy(50,14);cout<<"R= ";
75          cin>>opc;
76          switch(opc){
77              case 1:
78                  {
79                      hold=1;
80                      break;
81                  }
82              case 2:
83                  {
84                      hold=0;
85                      break;
86                  }
87              default:
88                  {
89                      cout<<"Ingrese una opcion valida"<<endl;
90                      break;
91                  }
92          }
93      }
94      return 0;
95  }
```

Ilustración 10 Función main

Main:

En la función main (función principal del código), se mandará a llamar la función PrincipioMenu, la cual contiene el diseño del menú. Se definen los apuntadores de las listas de listas en NULL y se realiza la lectura de cada una de ellas (listas de listas), esto para cargar los datos que están almacenados en archivos y evitar la repetitividad de los datos al momento de registrarlos. Además de tener el registro de todo lo realizado con anterioridad y guardar cambios posteriores para cuando se desee volver a ejecutar el código.

Se pide el ingreso del usuario y contraseña, y empieza a recorrerse el apuntador aux, el cual es tipo de Listas (listas de usuarios). Y hará un check-login, asegurándose del tipo de usuario que esta ingresando para ser redireccionado al menú correspondiente.

Si el booleano check es falso, procederá a pedirte un usuario que sea válido, de lo contrario terminará el programa.

Si el usuario y contraseña no se encuentran en los archivos, te regresará a un menú el cual te dará la opción de volver a intentar tu inicio de sesión o terminar el programa. De no seleccionar una de las opciones que ofrece el programa, te volverá a solicitar una opción válida.

validacion.h

```c
float validaFlotante(const char mensaje[]){
    int continuar=0;
    float flotante=0;

    do{
        printf("%s",mensaje);
        continuar=scanf("%f", &flotante);
        fflush(stdin);
    }while (continuar!=1);

    return flotante;
}

int validaEntero(const char mensaje[]){
    int continuar=0;
    int entero=0;

    do{
        printf("%s",mensaje);
        continuar=scanf("%d",&entero);
        fflush(stdin);
    }while (continuar!=1);

    return entero;
}
```

Ilustración 12 Validación flotante y entero

```c
void validaCadena(const char mensaje[], const char cadena[]){
    bool bandera = true;
    while(bandera){
        printf("%s", mensaje);
        scanf("%[^\n]",cadena);
        fflush(stdin);
        if(strlen(cadena)>29){
            continue;
        }else{
            for(int i=0; i<strlen(cadena); i++){
                if(isalpha(cadena[i])|| cadena[i]==' '){
                    if(i==(strlen(cadena)-1)){
                        bandera=false;
                    }
                }else{
                    break;
                }
            }
        }
    }
    return;
}
```

Ilustración 11Validación cadena

Validaciones:

La hoja de trabajo validacion.h se emplea una serie de validaciones para diferentes tipos de entradas, ya sea entero, flotante o una cadena. Estos se aseguran de que se ingrese el tipo de dato correcto en cada entrada (evita cadenas en enteros, por ejemplo).

archivos.h y archivosCli.h

Se describirán estas dos funciones a la par, ya que sus funciones similares. Ambas se encargarán de reservar espacios de memoria para las listas y generar las listas de listas de los usuarios y los clientes.

```
Usuario *registro(){
    decomain();
    int opc,hold=1;
    srand (time(NULL));
    Usuario *nuevo = new Usuario;
    nuevo->idUsuario=1000+rand()%(9999-1000)+1;
    gotoxy(45,5);
    cout<<"Nombre: ";
    cin>>nuevo->nombre;
    gotoxy(45,6);
    cout<<"Apellido: ";
    cin>>nuevo->apellido;
    gotoxy(45,7);
    nuevo->edad=validaEntero("Edad: ");
    gotoxy(45,8);
    cout<<"Telefono: ";
    cin>>nuevo->telefono;
    gotoxy(45,9);
    cout<<"Correo electronico: ";
    cin>>nuevo->correo;
    gotoxy(45,10);
    cout<<"Direccion: ";
    cin>>nuevo->dir;
    fflush(stdin);
    gotoxy(45,11);
    cout<<"Username: ";
    cin>>nuevo->username;
    gotoxy(45,12);
    cout<<"Password: ";
    cin>>nuevo->pass;
    nuevo->estado="Activo";
    while(hold==1){
        gotoxy(45,14);
        cout<<"Perfil: ";gotoxy(45,13);cout<<"1.-Administrador";gotoxy(45,14);cout<<"2.-Ejecutivo";gotoxy(45,15);cout<<"R= ";
        cin>>opc;
        gotoxy(40,28);
        cout<<"                                    ";
        switch(opc){
            case 1:
                {
                    nuevo->perfil=Administrador;
                    hold=0;
                    break;
                }
```

Ilustración 15 Lista usuario

```
                }
            case 2:
                {
                    nuevo->perfil=Ejecutivo;
                    hold=0;
                    break;
                }
            default:
                {
                    gotoxy(45,28);
                    cout<<"Ingrese una opcion valida"<<endl;
                    break;
                }
        }
    }

    nuevo->sig=NULL;
    return nuevo;
}
```

Ilustración 14 Lista Usuarios

```
Cliente *registroCli(){
    Cliente *nuevo = new Cliente;
    decomain();
    srand (time(NULL));
    fflush(stdin);
    nuevo->idCliente=1000+rand()%(9999-1000)+1;
    gotoxy(45,5);
    cout<<"Nombre: ";
    cin>>nuevo->nombrecli;
    gotoxy(45,6);
    cout<<"Apellido: ";
    fflush(stdin);
    cin>>nuevo->apellidocli;
    gotoxy(45,7);
    cout<<"Telefono: ";
    fflush(stdin);
    cin>>nuevo->telefonocli;
    gotoxy(45,8);
    cout<<"Referencia: ";
    fflush(stdin);
    cin>>nuevo->refecli;
    gotoxy(45,9);
    cout<<"Telefono de referencia: ";
    fflush(stdin);
    cin>>nuevo->tel_refecli;
    //----------------randtarjeta------------------
    generartarjeta();
    nuevo->tarjetacli=cardnumber;
    //---------------------------------------------

    makedeal(nuevo);
    nuevo->next=NULL;
    return nuevo;
}
```

Ilustración 13 Lista clientes

Listas:

Se declaran dos funciones del tipo de la estructura que se desean usar como listas. Estas funciones son “apuntadores” que tendrán una dirección de memoria y a su vez, guardarán los datos de cada una de ellas. Estas funciones guardan a su vez, el apuntador sig, que es el encargado de retener la dirección de memoria de su siguiente nodo, esto hace posible la utilización de las listas.

```
455  void agregarLista(){
456      srand (time(NULL));
457      Listas *nuevo = new Listas;
458
459      cout<<"Agregando lista a la LISTA"<<endl;
460      nuevo->idListas=10000+rand()%(99999-10000)+1;
461      int sigue=1;
462
463      primero=NULL;
464      ultimo=NULL;
465      while(sigue==1){
466          system("cls");
467          decomain();
468          agregarFinal();
469          gotoxy(45,26);
470          sigue = validaEntero("Tecla 1 para agregar otro: ");
471          system("cls");
472      }
473
474      nuevo->sig=NULL;
475      nuevo->lista=primero;
476      archivoEscritura(primero, nuevo->idListas);
477
478      if(primLista==NULL){
479          primLista=nuevo;
480          ultLista=nuevo;
481      }else{
482          ultLista->sig=nuevo;
483          ultLista=nuevo;
484
485      }
486  }
```

Ilustración 17 Función listas de listas usuario

```
761  void agregarListaCli(){
762      srand (time(NULL));
763      ListasClientes *nuevo = new ListasClientes;
764
765      nuevo->idListasCli=10000+rand()%(99999-10000)+1;
766      int sigue=1;
767
768      primerocli=NULL;
769      ultimocli=NULL;
770      while(sigue==1){
771          system("cls");
772          decomain();
773          agregarFinalCli();
774          gotoxy(45,26);
775          sigue = validaEntero("Tecla 1 para agregar otro: ");
776          system("cls");
777      }
778
779      nuevo->next=NULL;
780      nuevo->listacli=primerocli;
781      archivoEscrituraCli(primerocli, nuevo->idListasCli);
782
783      if(primListacli==NULL){
784          primListacli=nuevo;
785          ultListacli=nuevo;
786      }else{
787          ultListacli->next=nuevo;
788          ultListacli=nuevo;
789
790      }
791  }
```

Ilustración 16 Función listas de listas clientes

Listas de listas:

En las funciones agregarLista y agregarListaCli, se generan las listas de listas, una para los usuarios y otra para los clientes, el espacio de memoria que se reservara para cada una de estas listas se generará al inicio de las funciones (segunda línea de cada función), una vez reservado el espacio de memoria, se inicia un ciclo while que permitirá agregar datos a las listas hasta que el usuario desee terminar el proceso. Estas listas de listas guardaran la dirección de memoria del primer nodo de las listas.

```
void makedeal(Cliente *trato){
    int opc,hold=1;
    trato->deal.estadoContra=estado;

    while(hold==1){
        gotoxy(44,25);
        cout<<"                                                ";
        gotoxy(45,10);
        cout<<"Tipo de cliente:";gotoxy(45,11);cout<<"1.-Ahorrador";gotoxy(45,12);cout<<"2.-Deudor";gotoxy(45,13);cout
        cin>>opc;
        switch(opc){
            case 1:
                {
                    trato->deal.tipClien=Ahorrador;
                    hold=0;
                    break;
                }
            case 2:
                {
                    trato->deal.tipClien=DEudor;
                    hold=0;
                    break;
                }
            default:
                {
                    gotoxy(45,25);
                    cout<<"Inserte una opcion valida";
                    break;
                }
        }
    }

    opc=0;
    hold=1;
    gotoxy(45,14);
    cout<<"Fecha: ";
    cin>>trato->deal.fechaA;
    do{
        gotoxy(44,28);
        cout<<"                                                ";
        fflush(stdin);
        gotoxy(45,15);
        cout<<"Tipo de tarjeta del cliente: ";
        gotoxy(45,16);
        cout<<"1.-Ahorro";gotoxy(45,17);cout<<"2.-Credito";gotoxy(45,18);cout<<"3.-Debito";gotoxy(45,19);cout<<"4.-Cre
```

Ilustración 20 Función contrato

```
        cout<<"1.-Ahorro";gotoxy(45,17);cout<<"2.-Credito";gotoxy(45,18);cout<<"3.-Debito";gotoxy(45,1
        cin>>opc;
        switch(opc){
            case 1:
                {
                    trato->deal.prestamo=0;
                    trato->deal.tipotarcli=Ahorro;
                    gotoxy(45,21);
                    trato->deal.CanAhorro=validaFlotante("Cantidad a ahorrar: ");
                    trato->deal.interes=10;
                    gotoxy(45,22);
                    cout<<"Interes sobre ahorro es del 10% por mes";
                    trato->deal.anio=0.1*trato->deal.CanAhorro;
                    gotoxy(45,23);
                    cout<<"Cantidad acumulada dentro de un año: $"<<trato->deal.anio;
                    hold=0;
                    break;
                }
            case 2:
                {
                    trato->deal.CanAhorro=0;
                    trato->deal.tipotarcli=Credito;
                    fflush(stdin);
                    gotoxy(45,21);
                    trato->deal.prestamo=validaFlotante("Cantidad de prestamo: ");
                    trato->deal.interes=15;
                    gotoxy(45,22);
                    cout<<"Interes de credito es del 15% por mes";
                    trato->deal.anio=0.15*trato->deal.prestamo;
                    gotoxy(45,23);
                    cout<<"Cantidad de cobro anual por uso de la tarjeta: $"<<trato->deal.anio;
                    hold=0;
                    break;
                }
            case 3:
                {
                    trato->deal.tipotarcli=Debito;
                    hold=0;
                    break;
                }
            case 4:
                {
                    trato->deal.CanAhorro=0;
                    trato->deal.tipotarcli=Creditohipotecario;
                    gotoxy(45,21);
```

Ilustración 19 Función contrato

```
                    gotoxy(45,21);
                    cout<<"Que bien va a hipotecar el cliente?: ";
                    cin>>trato->deal.hipotecaref;
                    gotoxy(45,22);
                    trato->deal.prestamo=validaEntero("Valor del avaluo dado: ");
                    trato->deal.interes=20;
                    gotoxy(45,23);
                    cout<<"Interes de la hipoteca es del 20% por mes";
                    trato->deal.anio=0.20*trato->deal.prestamo;
                    gotoxy(45,24);
                    cout<<"Cantidad de cobro anual por la hipoteca: $"<<trato->deal.anio;
                    hold=0;
                    break;
                }
            default:
                {
                    gotoxy(45,28);
                    cout<<"Ingrese una opcion valida"<<endl;
                    break;
                }
        }
    }while(hold==1);
    getch();
    system("cls");
    decomain();
    fflush(stdin);
}
```

Ilustración 18 Función contrato

Contrato:

La función makedeal servirá para identificar el tipo de cliente (ahorrador o deudor) y generar un contrato, además de la fecha de registro, la función como parámetro la dirección de memoria del cliente para reservarle a este mismo su contrato correspondiente. Dependiendo de las características del contrato, tendrán un valor diferente de interés del ahorro, hipoteca, etc.

```
void agregarFinalCli(){
    Cliente *nuevo = registroCli();
    if(primerocli==NULL){
        primerocli=nuevo;
        ultimocli=nuevo;
    }
    else{
        ultimocli->next=nuevo;
        ultimocli=nuevo;
    }
}
```

Ilustración 22 Agregar final clientes

```
void agregarFinal(){
    Usuario *nuevo = registro();
    if(primero==NULL){
        primero=nuevo;
        ultimo=nuevo;
    }
    else{
        ultimo->sig=nuevo;
        ultimo=nuevo;
    }
}
```

Ilustración 21 Agregar final usuarios

Agregar Final:

Las funciones agregarFianl y agregarFinalCli, guardaran el nodo generado hasta al final de la lista, esta es la importancia de definir los 2 apuntadores extra en las estructuras, con esto identificamos el principio y final de cada lista.

```
void agregarAlistita(Usuario *primero){
    system("cls");
    decomain();
    struct Usuario *auxiliar=primero;
    bool bandera=true;
    int pos = 0;
    int i = 1;
    gotoxy(45,5);
    pos = validaEntero("Posición: ");
    if(pos==1){
        Usuario *nuevo = registro();
        nuevo->sig=primero;
        primero=nuevo;
    }else{
        while(auxiliar!=NULL){
            if(i == (pos-1) ){
                Usuario *nuevo = registro();
                nuevo->sig = auxiliar->sig;
                auxiliar->sig = nuevo;
                bandera=false;
                break;
            }else{
                auxiliar = auxiliar->sig;
                i++;
            }
        }
        if(bandera){
            gotoxy(30,28);
            cout<<"La posición NO existe, el nodo se agrega al final de la lista"<<endl;
            agregarFinal();
        }
    }
    getchar();
}
```

Ilustración 24 Agregar en posición usuarios

```
void agregarAlistitaCli(Cliente *primerocli){
    system("cls");
    decomain();
    struct Cliente *auxiliar=primerocli;
    bool bandera=true;
    int pos = 0;
    int i = 1;
    gotoxy(45,5);
    pos = validaEntero("Posicion: ");
    system("cls");
    if(pos==1){
        Cliente *nuevo = registroCli();
        nuevo->next=primerocli;
        primerocli=nuevo;
    }else{
        while(auxiliar!=NULL){
            if(i == (pos-1) ){
                Cliente *nuevo = registroCli();
                nuevo->next = auxiliar->next;
                auxiliar->next = nuevo;
                bandera=false;
                break;
            }else{
                auxiliar = auxiliar->next;
                i++;
            }
        }
        if(bandera){
            gotoxy(30,28);
            cout<<"La posición NO existe, el nodo se agrega al final de la lista"<<endl;
            agregarFinalCli();
        }
    }
    getchar();
}
```

Ilustración 23 Agregar en posición clientes

Agregar nodos en posición:

Las funciones agregarAlistita y agregarAlistitaCli, guardaran un nodo en la posición deseada por el usuario, estos reciben como parámetro la dirección de memoria del tipo de lista que se desea agregar. A su vez, se genera un auxiliar que servirá para recorrer las listas, y un contador para saber en que punto la lista se va a agregar.

```
void archivoEscrituraCli(Cliente *primerocli,int id){
    Cliente *aux=primerocli;
    char arch[20];
    itoa(id,arch,10);
    strcat(arch,".xls");
    ofstream archivo(arch);

    while(aux!=NULL){
        archivo<<aux->idCliente<<"\t";
        archivo<<aux->nombrecli<<"\t";
        archivo<<aux->apellidocli<<"\t";
        archivo<<aux->telefonocli<<"\t";
        archivo<<aux->refecli<<"\t";
        archivo<<aux->tel_refecli<<"\t";
        archivo<<aux->tarjetacli<<"\t";

        archivo<<aux->deal.tipotarcli<<"\t";
        archivo<<aux->deal.estadoContra<<"\t";
        archivo<<aux->deal.fechaA<<"\t";
        archivo<<aux->deal.tipClien<<"\t";
        archivo<<aux->deal.CanAhorro<<"\t";
        archivo<<aux->deal.prestamo<<"\t";
        archivo<<aux->deal.interes<<"\t";
        archivo<<aux->deal.anio<<"\t";
        archivo<<aux->deal.hipotecaref<<endl;

        ultimocli=aux;
        aux=aux->next;
    }

    archivo.close();
}
```

Ilustración 26 Escritura de listas clientes

```
void archivoEscritura(Usuario *primero,int id){
    Usuario *aux=primero;
    char arch[20];
    itoa(id,arch,10);
    strcat(arch,".xls");
    ofstream archivo(arch);

    while(aux!=NULL){
        archivo<<aux->idUsuario<<"\t";
        archivo<<aux->nombre<<"\t";
        archivo<<aux->apellido<<"\t";
        archivo<<aux->edad<<"\t";
        archivo<<aux->telefono<<"\t";
        archivo<<aux->correo<<"\t";
        archivo<<aux->dir<<"\t";
        archivo<<aux->username<<"\t";
        archivo<<aux->pass<<"\t";
        archivo<<aux->estado<<"\t";
        archivo<<aux->perfil<<endl;
        ultimo=aux;
        aux=aux->sig;
    }
    archivo.close();
}
```

Ilustración 25 Escritura de listas usuarios

Archivo estructura:

Las funciones de archivo escritura, se encargarán de generar un archivo tipo .xls (Excel) para guardar los datos registrados dependiendo del tipo de lista que se haya generado.
De igual forma reciben las direcciones de memoria del tipo de lista y el id, que se genera aleatoriamente para generar un archivo con el nombre de ese id, a su vez, recorren toda la lista y sobrescriben los datos generados en la ejecución del programa.

```
void modificalistita(Usuario *primero, int estado,int aa){
    struct Usuario *auxiliar=primero;
    int folmod,op,hold=1,mod=1,bb=aa+3,opc;

    gotoxy(45,aa+3);
    cout<<"ID's de la lista seleccionada";
    while(primero!=NULL){
        bb+=1;
        gotoxy(45,bb);
        cout<<primero->idUsuario<<" -----> "<<primero->nombre;
        primero=primero->sig;
    }

        while(mod==1){
        gotoxy(45,bb+1);
        folmod=validaEntero("Selecciona el ID del Usuario a modificar: ");
        gotoxy(3,28);
        cout<<"                                                ";
        bool bandera = true;
        while(bandera){
        primero=auxiliar;
            while(primero!=NULL){
                if(folmod != primero->idUsuario){
                    primero=primero->sig;
                    bandera=false;
                }else{
                    bandera = true;
                    break;
                }
            }
            if(bandera==false){
                gotoxy(3,28);
                cout<<"El ID no esta registrado"<<endl;
            }else{
                bandera=false;
                mod=2;
            }
        }
    }
    system("cls");
    decomain();
    hold=1;
    while(auxiliar!=NULL){
        if(auxiliar->idUsuario==folmod){
```

Ilustración 30 Modificar lista usuario

```
        if(auxiliar->idUsuario==folmod){
            if(estado==1){
                gotoxy(45,3);
                cout<<"Nombre: ";
                cin>>auxiliar->nombre;
                gotoxy(45,4);
                cout<<"Apellido: ";
                cin>>auxiliar->apellido;
                gotoxy(45,5);
                auxiliar->edad=validaEntero("Edad: ");
                gotoxy(45,6);
                cout<<"Telefono: ";
                cin>>auxiliar->telefono;
                gotoxy(45,7);
                cout<<"Correo electronico: ";
                cin>>auxiliar->correo;
                gotoxy(45,8);
                cout<<"Direccion: ";
                cin>>auxiliar->dir;
                gotoxy(45,9);
                cout<<"Username: ";
                cin>>auxiliar->username;
                gotoxy(45,10);
                cout<<"Password: ";
                cin>>auxiliar->pass;

                while(hold==1){
                gotoxy(45,12);
                cout<<"Perfil: ";gotoxy(45,13);cout<<"1.-Administrador";gotoxy(45,14);cout<<"2.-Ejecutivo";gotoxy(45,15);cout<<"R= ";
                cin>>opc;
                gotoxy(40,28);
                cout<<"                                                ";
                switch(opc){
                    case 1:
                        {
                            auxiliar->perfil=Administrador;
                            hold=0;
                            break;
                        }
                    case 2:
                        {
                            auxiliar->perfil=Ejecutivo;
                            hold=0;
                            break;
                        }
```

Ilustración 29 Modificar lista usuario

```
void modificalistitaCli(Cliente *primerocli, int estado,int aa){
    struct Cliente *auxiliar=primerocli;
    int folmod,op,hold=1,mod=1,bb=aa+3;
    gotoxy(45,aa+3);
    cout<<"ID's de la lista seleccionada"<<endl;
    while(primerocli!=NULL){
        bb+=1;
        gotoxy(45,bb);
        cout<<primerocli->idCliente<<" -----> "<<primerocli->nombrecli<<endl;
        primerocli=primerocli->next;
    }
    //--------------------------------------------------------------------------
    while(mod==1){
        gotoxy(45,bb+1);
        folmod=validaEntero("Selecciona el ID del Usuario a modificar: ");
        gotoxy(45,28);
        cout<<"                                                ";
        bool bandera = true;
        while(bandera){
        primerocli=auxiliar;
            while(primerocli!=NULL){
                if(folmod != primerocli->idCliente){
                    primerocli=primerocli->next;
                    bandera=false;
                }else{
                    bandera = true;
                    break;
                }
            }
            if(bandera==false){
                gotoxy(45,28);
                cout<<"El ID no esta registrado"<<endl;
            }else{
                bandera=false;
                mod=2;
            }
        }
    }
    //--------------------------------------------------------------------------
    system("cls");
    decomain();
    primerocli=auxiliar;
    int opc;
    hold=1;
```

Ilustración 28 Modificar lista cliente

```
    while(auxiliar!=NULL){
        if(auxiliar->idCliente==folmod){
            if(estado==1){
                gotoxy(45,3);
                auxiliar->telefonocli=validaEntero("Telefono Cliente: ");
                gotoxy(45,4);
                cout<<"Referencia: ";
                cin>>auxiliar->refecli;
                gotoxy(45,5);
                auxiliar->tel_refecli=validaEntero("Telefono Referencia: ");
                gotoxy(45,6);
                while(hold==1){
                    gotoxy(45,7);
                    cout<<"Tipo de cliente:";gotoxy(45,8);cout<<"1.-Ahorrador";gotoxy(45,9);cout<<
                    cin>>opc;
                    switch(opc){
                        case 1:
                            {
                                auxiliar->deal.tipClien=Ahorrador;
                                hold=0;
                                break;
                            }
                        case 2:
                            {
                                auxiliar->deal.tipClien=DEudor;
                                hold=0;
                                break;
                            }
                        default:
                            {
                                break;
                            }
                    }
                }
            }
            if(estado==2){
                        }
                    default:
                        {                                                    toxy(45,5);cout<<
                            gotoxy(45,28);
                            cout<<"Ingrese una opcion valida"<<endl;
                            break;
                        }
                }
            }
        }
```

Ilustración 27 Modificar lista cliente

```
                cout<<"Estado del usuario: ";
                cin>>auxiliar->estado;
            }
        }
        auxiliar=auxiliar->sig;
    }
    system("cls");
}
```

```
                                hold=0;
                                break;
                            }
                        case 2:
                            {
                                auxiliar->deal.estadoContra=Inactivo;
                                hold=0;
                                break;
                            }
                        case 3:
                            {
                                auxiliar->deal.estadoContra=En_Deuda;
                                hold=0;
                                break;
                            }
                        case 4:
                            {
                                auxiliar->deal.estadoContra=Cancelado;
                                hold=0;
                                break;
                            }
                        case 5:
                            {
                                auxiliar->deal.estadoContra=Sin_Deuda;
                                hold=0;
                                break;
                            }
                        default:
                            {
                                break;
                            }
                    }
                }
            }
            auxiliar=auxiliar->next;
        }
        system("cls");
    }
```

Ilustración 31 Modificar lista cliente

Modificar lista:

Las funciones de modificar lista, recibe como parámetro, la dirección de memoria del primer nodo de la lista correspondiente. En estas te solicitan el ID de la lista de listas que quieres modificar y se recorrerá en las listas de listas hasta encontrar el ID solicitado, una vez seleccionada la lista, procederá a pedirte el ID del usuario que deseas modificar y al igual que con las listas de listas, se recorrerá la dirección de memoria hasta encontrar el ID solicitado, una vez identificado el nodo que se desea modificar, volverá a solicitar los nuevos datos que serán sobrescritos en el nodo.

```
Usuario *archivoLectura(int id){
    Usuario *primero=NULL;
    Usuario *ultimo=NULL;
    Usuario *nuevo=NULL;

    char archivo[20];
    itoa(id,archivo,10);
    strcat(archivo,".xls");

    ifstream arch(archivo);
    string linea, c;

    cout<<id<<endl;
    if(arch.fail()) cerr<<"Error al abrir el archivo"<<endl;
    else{
        while(getline(arch,linea)){
            nuevo = new Usuario;
            stringstream lee(linea);

            lee>>nuevo->idUsuario;
            getline(lee,c,'\t');
            getline(lee,nuevo->nombre,'\t');
            getline(lee,nuevo->apellido,'\t');
            lee>>nuevo->edad;
            getline(lee,c,'\t');
            lee>>nuevo->telefono;
            getline(lee,c,'\t');
            getline(lee,nuevo->correo,'\t');
            getline(lee,nuevo->dir,'\t');
            getline(lee,nuevo->username,'\t');
            getline(lee,nuevo->pass,'\t');
            getline(lee,nuevo->estado,'\t');
            getline(lee,nuevo->perfil,'\n');

            nuevo->sig=NULL;
            if(primero==NULL){
                primero = nuevo;
                ultimo=nuevo;
            }else{
                ultimo->sig=nuevo;
                ultimo=nuevo;
            }
        }
    }
    arch.close();
```

Ilustración 33 Lectura de archivos usuario

```
Cliente *archivoLecturaCli(int id){
    Cliente *primerocli=NULL;
    Cliente *ultimocli=NULL;
    Cliente *nuevo=NULL;

    char archivo[20];
    itoa(id,archivo,10);
    strcat(archivo,".xls");

    ifstream arch(archivo);
    string linea, c;

    cout<<id<<endl;
    if(arch.fail()) cerr<<"Error al abrir el archivo"<<endl;
    else{
        while(getline(arch,linea)){
            nuevo = new Cliente;
            stringstream lee(linea);

            lee>>nuevo->idCliente;
            getline(lee,c,'\t');
            getline(lee,nuevo->nombrecli,'\t');
            getline(lee,nuevo->apellidocli,'\t');
            lee>>nuevo->telefonocli;
            getline(lee,c,'\t');
            getline(lee,nuevo->refecli,'\t');
            lee>>nuevo->tel_refecli;
            getline(lee,c,'\t');
            lee>>nuevo->tarjetacli;
            getline(lee,c,'\t');

            getline(lee,nuevo->deal.tipotarcli,'\t');
            getline(lee,nuevo->deal.estadoContra,'\t');
            getline(lee,nuevo->deal.fechaA,'\t');
            getline(lee,nuevo->deal.tipClien,'\t');

            lee>>nuevo->deal.CanAhorro;
            getline(lee,c,'\t');
            lee>>nuevo->deal.prestamo;
            getline(lee,c,'\t');
            lee>>nuevo->deal.interes;
            getline(lee,c,'\t');
            lee>>nuevo->deal.anio;
            getline(lee,c,'\t');
            getline(lee,nuevo->deal.hipotecaref,'\n');
```

Ilustración 32 Lectura de archivos cliente

```
478                 nuevo->next=NULL;
479
480                 if(primerocli==NULL){
481                     primerocli= nuevo;
482                     ultimocli=nuevo;
483                 }else{
484                     ultimocli->next=nuevo;
485                     ultimocli=nuevo;
486                 }
487             }
488         }
489         arch.close();
490         return primerocli;
491     }
```

Ilustración 34 Lectura archivos cliente

Archivo lectura:

Las funciones de archivo lectura servirán para recopilar los datos de los archivos generados de las listas. Estas abren el archivo y obtienen las líneas separadas por tabulaciones, una vez llegado al ultimo dato de la lista, se lee un salto de línea y continua con el siguiente registro. Estás lecturas cambiaran dependiendo de la cantidad de archivos generados. Se declaran al inicio del programa, esto para que haya datos existentes en las listas y se puedan seguir añadiendo más registros o modificar los ya existentes.

```
504 void escrituraListaClitxt(){
505     ListasClientes *aux=primListacli;
506
507     ofstream archivo("ListaDeListasCli.txt");
508     while(aux!=NULL){
509         archivo<<aux->idListasCli<<"\n";
510         aux=aux->next;
511     }
512     archivo.close();
513 }
```

Ilustración 36 Escritura lista de listas cliente

```
360 void escrituraListatxt(){
361     Listas *aux=primLista;
362
363     ofstream archivo("ListaDeusuarios.txt");
364     while(aux!=NULL){
365         archivo<<aux->idListas<<"\n";
366         aux=aux->sig;
367     }
368     archivo.close();
369 }
```

Ilustración 35 Escritura lista de listas usuario

Escritura de listas:

Los archivos de escritura de lista de listas, guardan en un archivo .txt (Texto/Bloc de notas), los id de las listas de listas, estas son necesarias al momento de la lectura para identificar a cual lista de listas pertenecen las listas generadas.

```
547  void mostrar_listaCli(Cliente *primerocli){
548      struct Cliente *auxiliar=primerocli;
549      cout<<" "<<endl;
550      SetConsoleTextAttribute(hd, 12);
551  //  cout<<"Ubicacion"<<setw(10);
552      cout<<"--ID--"<<setw(10);
553      cout<<"Nombre"<<setw(15);
554      cout<<"Apellido"<<setw(20);
555      cout<<"Telefono"<<setw(15);
556      cout<<"Referencia"<<setw(15);
557      cout<<"Tel Ref"<<setw(21);
558      cout<<"No. de tarjeta"<<setw(15);
559      cout<<"Est Contrat"<<endl;
560
561      SetConsoleTextAttribute(hd, 9);
562      while(auxiliar!=NULL){
563          cout<<auxiliar->idCliente<<setw(11);
564          cout<<auxiliar->nombrecli<<setw(15);
565          cout<<auxiliar->apellidocli<<setw(20);
566          cout<<auxiliar->telefonocli<<setw(15);
567          cout<<auxiliar->refecli<<setw(15);
568          cout<<auxiliar->tel_refecli<<setw(21);
569          cout<<auxiliar->tarjetacli<<setw(15);
570          cout<<auxiliar->deal.estadoContra<<endl;
571
572          auxiliar=auxiliar->next;
573      }
574      cout<<" "<<endl;
575  }
576
577  void checarcontrato(Cliente *primerocli){
578      struct Cliente *auxiliar=primerocli;
579      cout<<" "<<endl;
580      SetConsoleTextAttribute(hd, 12);
581      cout<<"--ID--"<<setw(10);
582      cout<<"Nombre"<<setw(15);
583      cout<<"Fecha"<<setw(20);
584      cout<<"No. de tarjeta"<<setw(20);
585      cout<<"Tip Tarjeta"<<setw(12);
586      cout<<"Prestamo"<<setw(15);
587      cout<<"Est Contrato"<<setw(15);
588      cout<<"Tip Cliente"<<setw(10);
589      cout<<"Interes"<<setw(12);
590      cout<<"IGA"<<setw(10);          //Ingreso generado en un año
591      cout<<"DGA"<<setw(10);          //Deuda generada en un año
```

Ilustración 38 Mostrar lista de clientes

```
592      cout<<"Hipoteca"<<endl;
593
594      SetConsoleTextAttribute(hd, 9);
595      while(auxiliar!=NULL){
596          cout<<auxiliar->idCliente<<setw(10);
597          cout<<auxiliar->nombrecli<<setw(15);
598          cout<<auxiliar->deal.fechaA<<setw(20);
599          cout<<auxiliar->tarjetacli<<setw(20);
600          cout<<auxiliar->deal.tipotarcli<<setw(12);
601          cout<<auxiliar->deal.prestamo<<setw(15);
602          cout<<auxiliar->deal.estadoContra<<setw(15);
603          cout<<auxiliar->deal.tipClien<<setw(10);
604          cout<<auxiliar->deal.interes<<"%"<<setw(12);
605          if((auxiliar->deal.tipClien==DEudor)||(auxiliar->deal.tipClien==deudOR)){
606              cout<<" "<<setw(10);
607              cout<<auxiliar->deal.anio<<setw(10);
608          }else{
609              cout<<auxiliar->deal.anio<<setw(10);
610              cout<<" "<<setw(10);
611          }
612          cout<<auxiliar->deal.hipotecaref<<endl;
613          auxiliar=auxiliar->next;
614      }
615
616      cout<<" "<<endl;
617  }
```

Ilustración 37 Mostrar lista de clientes

```
void mostrar_listaCli(Cliente *primerocli){
    struct Cliente *auxiliar=primerocli;
    cout<<" "<<endl;
    SetConsoleTextAttribute(hd, 12);
//  cout<<"Ubicacion"<<setw(10);
    cout<<"--ID--"<<setw(10);
    cout<<"Nombre"<<setw(15);
    cout<<"Apellido"<<setw(20);
    cout<<"Telefono"<<setw(15);
    cout<<"Referencia"<<setw(15);
    cout<<"Tel Ref"<<setw(21);
    cout<<"No. de tarjeta"<<setw(15);
    cout<<"Est Contrat"<<endl;

    SetConsoleTextAttribute(hd, 9);
    while(auxiliar!=NULL){
        cout<<auxiliar->idCliente<<setw(11);
        cout<<auxiliar->nombrecli<<setw(15);
        cout<<auxiliar->apellidocli<<setw(20);
        cout<<auxiliar->telefonocli<<setw(15);
        cout<<auxiliar->refecli<<setw(15);
        cout<<auxiliar->tel_refecli<<setw(21);
        cout<<auxiliar->tarjetacli<<setw(15);
        cout<<auxiliar->deal.estadoContra<<endl;

        auxiliar=auxiliar->next;
    }
    cout<<" "<<endl;
}
```

Ilustración 42 Mostrar lista de clietnes

```
void mostrar_lista(Usuario *primero){
    struct Usuario *auxiliar=primero;
    cout<<" \n"<<endl;
    cout<<"Mostrando la lista completa"<<endl;
    SetConsoleTextAttribute(hd, 12);
    cout<<"--ID--"<<setw(11);
    cout<<"Nombre"<<setw(15);
    cout<<"Apellido"<<setw(11);
    cout<<"Edad"<<setw(15);
    cout<<"Telefono"<<setw(20);
    cout<<"Correo"<<setw(20);
    cout<<"Direccion"<<setw(20);
    cout<<"Username"<<setw(15);
    cout<<"Password"<<setw(15);
    cout<<"Estado"<<setw(13);
    cout<<"Perfil"<<endl;

    SetConsoleTextAttribute(hd, 9);
    while(auxiliar!=NULL){
        cout<<auxiliar->idUsuario<<setw(11);
        cout<<auxiliar->nombre<<setw(15);
        cout<<auxiliar->apellido<<setw(11);
        cout<<auxiliar->edad<<setw(15);
        cout<<auxiliar->telefono<<setw(20);
        cout<<auxiliar->correo<<setw(20);
        cout<<auxiliar->dir<<setw(20);
        cout<<auxiliar->username<<setw(15);
        cout<<auxiliar->pass<<setw(15);
        cout<<auxiliar->estado<<setw(13);
        cout<<auxiliar->perfil<<endl;
        auxiliar=auxiliar->sig;
    }
}

//Mostrar LISTOTA
void mostrar_listas(){
    system("cls");
    struct Listas *aux=primLista;

    aux=primLista;
    while(aux!=NULL){
        SetConsoleTextAttribute(hd, 10);
        cout<<endl<<"ID de la lista: "<<aux->idListas;
        mostrar_lista(aux->lista);
        aux=aux->sig;
    }
    getch();
}
```

Ilustración 41 Mostrar lista de usuarios

```
void mostrar_listasCli(){
    system("cls");
    struct ListasClientes *aux=primListacli;
    int opc,hold=1,aa=8;

    aux=primListacli;
    while(aux!=NULL){
        SetConsoleTextAttribute(hd, 10);
        cout<<endl<<"ID de la lista: "<<aux->idListasCli;
        mostrar_listaCli(aux->listacli);
        aa+=5;
        aux=aux->next;
    }

    do{
        gotoxy(40,aa+2);
        SetConsoleTextAttribute(hd, 14);
        cout<<"1.-Consultar contratos";gotoxy(40,aa+3);cout<<"2.-Filtrar por Ahorradores";gotoxy(40,aa+4);cout<<"3.-Filtrar por Deudores";gotoxy(40,aa+5);cout<<"0.-Salir";go
        cin>>opc;
        aa=8;
        system("cls");
        switch(opc){
            case 1:
                {
                    aux=primListacli;
                    while(aux!=NULL){
                        SetConsoleTextAttribute(hd, 10);
                        cout<<endl<<"ID lista: "<<aux->idListasCli;
                        checarcontrato(aux->listacli);
                        aa+=7;
                        aux=aux->next;
                    }
                    break;
                }
            case 2:
                {
                    aux=primListacli;
                    while(aux!=NULL){
                        SetConsoleTextAttribute(hd, 10);
                        cout<<endl<<"ID lista: "<<aux->idListasCli;
                        filtrar(aux->listacli,1);
                        aa+=7;
                        aux=aux->next;
                    }
                    break;
                }
```

Ilustración 40 Mostrar lista de clientes

```
                {
                    aux=primListacli;
                    while(aux!=NULL){
                        SetConsoleTextAttribute(hd, 10);
                        cout<<endl<<"ID lista: "<<aux->idListasCli;
                        checarcontrato(aux->listacli);
                        aa+=7;
                        aux=aux->next;
                    }
                    break;
                }
            case 2:
                {
                    aux=primListacli;
                    while(aux!=NULL){
                        SetConsoleTextAttribute(hd, 10);
                        cout<<endl<<"ID lista: "<<aux->idListasCli;
                        filtrar(aux->listacli,1);
                        aa+=7;
                        aux=aux->next;
                    }
                    break;
                }
            case 3:
                {
                    aux=primListacli;
                    while(aux!=NULL){
                        SetConsoleTextAttribute(hd, 10);
                        cout<<endl<<"ID lista: "<<aux->idListasCli;
                        filtrar(aux->listacli,2);
                        aa+=7;
                        aux=aux->next;
                    }
                    break;
                }
            case 0:
                {
                    hold=0;
                    break;
                }
            default:
                {
                    cout<<"Ingrese una opcion valida"<<endl;
                    break;
                }
        }
    }while(hold==1);
}
```

Ilustración 39 Mostrar lista de clientes

```
void filtrar(Cliente *primerocli,int DEAH){
    struct Cliente *auxiliar=primerocli;
    cout<<" "<<endl;
    SetConsoleTextAttribute(hd, 12);
    cout<<"--ID--"<<setw(10);
    cout<<"Nombre"<<setw(15);
    cout<<"Fecha"<<setw(20);
    cout<<"No. de tarjeta"<<setw(20);
    cout<<"Tip Tarjeta"<<setw(12);
    cout<<"Prestamo"<<setw(15);
    cout<<"Est Contrato"<<setw(15);
    cout<<"Tip Cliente"<<setw(10);
    cout<<"Interes"<<setw(12);
    cout<<"IGA"<<setw(10);          //Ingreso generado en un año
    cout<<"DGA"<<setw(10);          //Deuda generada en un año
    cout<<"Hipoteca"<<endl;
    if(DEAH==2){
        if(auxiliar->deal.tipClien==deudOR or auxiliar->deal.tipClien==DEudor){
            SetConsoleTextAttribute(hd, 9);
            while(auxiliar!=NULL){
                cout<<auxiliar->idCliente<<setw(10);
                cout<<auxiliar->nombrecli<<setw(15);
                cout<<auxiliar->deal.fechaA<<setw(20);
                cout<<auxiliar->tarjetacli<<setw(20);
                cout<<auxiliar->deal.tipotarcli<<setw(12);
                cout<<auxiliar->deal.prestamo<<setw(15);
                cout<<auxiliar->deal.estadoContra<<setw(15);
                cout<<auxiliar->deal.tipClien<<setw(10);
                cout<<auxiliar->deal.interes<<"%"<<setw(12);
                if((auxiliar->deal.tipClien==DEudor)||(auxiliar->deal.tipClien==deudOR)){
                    cout<<" "<<setw(10);
                    cout<<auxiliar->deal.anio<<setw(10);
                }else{
                    cout<<auxiliar->deal.anio<<setw(10);
                    cout<<" "<<setw(10);
                }
                cout<<auxiliar->deal.hipotecaref<<endl;
                auxiliar=auxiliar->next;
            }
            cout<<" "<<endl;
        }
    }
    if(DEAH==1){
        if(auxiliar->deal.tipClien==Ahorrador or auxiliar->deal.tipClien==ahorradOR){
            SetConsoleTextAttribute(hd, 9);
            while(auxiliar!=NULL){
                cout<<auxiliar->idCliente<<setw(10);
                cout<<auxiliar->nombrecli<<setw(15);
```

Ilustración 45 Filtrar clientes

```
    if(DEAH==1){
        if(auxiliar->deal.tipClien==Ahorrador or auxiliar->deal.tipClien==ahorradOR){
            SetConsoleTextAttribute(hd, 9);
            while(auxiliar!=NULL){
                cout<<auxiliar->idCliente<<setw(10);
                cout<<auxiliar->nombrecli<<setw(15);
                cout<<auxiliar->deal.fechaA<<setw(20);
                cout<<auxiliar->tarjetacli<<setw(20);
                cout<<auxiliar->deal.tipotarcli<<setw(12);
                cout<<auxiliar->deal.prestamo<<setw(15);
                cout<<auxiliar->deal.estadoContra<<setw(15);
                cout<<auxiliar->deal.tipClien<<setw(10);
                cout<<auxiliar->deal.interes<<"%"<<setw(12);
                if((auxiliar->deal.tipClien==DEudor)||(auxiliar->deal.tipClien==deudOR)){
                    cout<<" "<<setw(10);
                    cout<<auxiliar->deal.anio<<setw(10);
                }else{
                    cout<<auxiliar->deal.anio<<setw(10);
                    cout<<" "<<setw(10);
                }
                cout<<auxiliar->deal.hipotecaref<<endl;
                auxiliar=auxiliar->next;
            }
            cout<<" "<<endl;
        }
    }
}
```

Ilustración 44 Filtrar clientes

```
void dirCli(Cliente *primerocli){
    struct Cliente *auxiliar=primerocli;
    cout<<" "<<endl;
    SetConsoleTextAttribute(hd, 12);
    cout<<"Ubicacion"<<setw(10);
    cout<<"--ID--"<<setw(10);
    cout<<"Nombre"<<setw(15);
    cout<<"Apellido"<<setw(20);
    cout<<"Telefono"<<setw(15);
    cout<<"Referencia"<<setw(15);
    cout<<"Tel Ref"<<setw(21);
    cout<<"No. de tarjeta"<<setw(15);
    cout<<"Est Contrat"<<setw(15);
    cout<<"Siguiente"<<endl;

    SetConsoleTextAttribute(hd, 9);
    while(auxiliar!=NULL){
        cout<<auxiliar<<setw(10);
        cout<<auxiliar->idCliente<<setw(11);
        cout<<auxiliar->nombrecli<<setw(15);
        cout<<auxiliar->apellidocli<<setw(20);
        cout<<auxiliar->telefonocli<<setw(15);
        cout<<auxiliar->refecli<<setw(15);
        cout<<auxiliar->tel_refecli<<setw(21);
        cout<<auxiliar->tarjetacli<<setw(15);
        cout<<auxiliar->deal.estadoContra<<setw(15);
        cout<<auxiliar->next<<endl;
        auxiliar=auxiliar->next;
    }

    cout<<endl<<"Primer nodo"<<primerocli;
    cout<<endl<<"Ultimo nodo"<<ultimocli;
    cout<<" "<<endl;
}
```

Ilustración 46 Direcciones de clientes

```
void Direccionescli(){
    system("cls");
    struct ListasClientes *aux=primListacli;
    cout<<"Ubicacion Lista"<<setw(20);
    cout<<"Lista(Pimer nodo)"<<setw(20);
    cout<<"Siguiente"<<endl;

    while(aux!=NULL){
        cout<<aux<<setw(20);
        cout<<aux->listacli<<setw(20);
        cout<<aux->next<<endl;
        aux=aux->next;
    }
    cout<<endl<<"Primer LISTA"<<primListacli;
    cout<<endl<<"Ultima LISTA"<<ultListacli;

    aux=primListacli;
    while(aux!=NULL){
        SetConsoleTextAttribute(hd, 10);
        cout<<endl<<"ID de la lista: "<<aux->idListasCli;
        dirCli(aux->listacli);
        aux=aux->next;
    }
    getch();
    system("cls");
}
```

Ilustración 43 Direcciones listas de clientes

```
488 void direccioneschiquitas(Usuario *primero){
489     struct Usuario *auxiliar=primero;
490     cout<<" \n"<<endl;
491     SetConsoleTextAttribute(hd, 12);
492     cout<<"Ubicacion"<<setw(10);
493     cout<<"--ID--"<<setw(11);
494     cout<<"Nombre"<<setw(15);
495     cout<<"Estado"<<setw(13);
496     cout<<"Perfil"<<setw(20);
497     cout<<"Siguiente"<<endl;
498 
499     SetConsoleTextAttribute(hd, 9);
500     while(auxiliar!=NULL){
501         cout<<auxiliar<<setw(10);
502         cout<<auxiliar->idUsuario<<setw(11);
503         cout<<auxiliar->nombre<<setw(15);
504         cout<<auxiliar->estado<<setw(13);
505         cout<<auxiliar->perfil<<setw(20);
506         cout<<auxiliar->sig<<endl;
507         auxiliar=auxiliar->sig;
508     }
509 
510     cout<<endl<<"Primer nodo"<<primero;
511     cout<<endl<<"Ultimo nodo"<<ultimo;
512 
513     getch();
514     system("cls");
515 }
```

Ilustración 47 Direcciones usuario

```
516 void direcciones(){
517     system("cls");
518     struct Listas *aux=primLista;
519     cout<<"Ubicacion Lista"<<setw(20);
520     cout<<"Lista(Pimer nodo)"<<setw(20);
521     cout<<"Siguiente"<<endl;
522 
523     while(aux!=NULL){
524         cout<<aux<<setw(20);
525         cout<<aux->lista<<setw(20);
526         cout<<aux->sig<<endl;
527         aux=aux->sig;
528     }
529     cout<<endl<<"Primer LISTA"<<primLista;
530     cout<<endl<<"Ultima LISTA"<<ultLista;
531 
532     aux=primLista;
533     while(aux!=NULL){
534         cout<<endl<<"ID de la lista: "<<aux->idListas;
535         direccioneschiquitas(aux->lista);
536         aux=aux->sig;
537     }
538 }
```

Ilustración 48 Direcciones listas usuarios

Impresión de listas:

Las funciones mostradas anteriormente son impresiones de listas y listas de listas, mostrando todos los datos por tipo de lista, la función filtrar, muestra los datos en base a la opción que selecciones. Estas funciones inician desde la primera dirección de memoria de las listas, imprimen, y una vez abada la impresión de esa lista, avanza su dirección de memoria y realiza una nueva impresión. Las impresiones de las listas son separadas por el ID de listas de listas de correspondiente.

```
1  int generartarjeta(){
2      int i;
3      srand (time(NULL));
4      cardnumber = rand() % 9 + 1;
5      for (i = 0; i < 15; i++)
6      {
7          cardnumber *= 10;
8          cardnumber += rand() % 10;
9      }
10     return cardnumber;
11 }
```

Ilustración 49 Número de tarjeta aleatorio

Número de tarjeta:

Esta función generara un número de tarjeta aleatorio, esto para ahorrar al usuario de generar los números de tarjeta manualmente. Al ser una función tipo entero, regresara un valor, en este regresa el número de tarjeta ya generado.

```
void menuadmin(string name){
    system("cls");
    int op=1;
    decomain();
    gotoxy(40,10);
    cout<<"--->Bienvenido Aministrador "<<name<<"<---";
    while(op!=0){
        decomain();
        gotoxy(40,12);
        cout<<"1.- Consultar Listas";gotoxy(40,13);cout<<"2.-Agregar Nueva lis
        gotoxy(40,18);
        op = validaEntero("Que desea hacer: ");
        switch(op){
            case 1:
                {
                    system("cls");
                    mostrar_listas();
                    system("cls");
                    break;
                }
            case 2:
                {
                    system("cls");
                    decomain();
                    agregarLista();
                    escrituraListatxt();
                    break;
                }
            case 3:
                {
                    system("cls");
                    ModificarList(1,1);
                    break;
                }
            case 4:
                {
                    system("cls");
                    ModificarList(1,2);
                    break;
                }
```

Ilustración 53 Menú administrador

```
                    break;
                }
            case 5:
                {
                    system("cls");
                    ModificarList(2,1);
                    break;
                }
            case 0:
                {
                    system("cls");
                    break;
                }
            case 2428:
                {
                    system("cls");
                    direcciones();
                    break;
                }
            default:
                {
                    cout<<"Ingrese una opcion valida"<<endl;
                }
        }

    }
}
```

Ilustración 52 Menú administrador

```
void menuejec(string name){
    system("cls");
    int op=1;
    decomain();
    gotoxy(40,10);
    cout<<"--->Bienvenido Ejecutivo "<<name<<"<---";
    while(op!=0){
        decomain();
        gotoxy(40,12);
        cout<<"1.- Consultar Listas de clientes";gotoxy(40,13);cout<<
        gotoxy(40,18);
        op = validaEntero("Que desea hacer: ");
        switch(op){
            case 1:
                {
                    system("cls");
                    mostrar_listasCli();
                    system("cls");
                    break;
                }
            case 2:
                {
                    system("cls");
                    agregarListaCli();
                    escrituraListaClitxt();
                    break;
                }
            case 3:
                {
                    system("cls");
                    ModificarListCli(1,1);
                    break;
                }
            case 4:
                {
                    system("cls");
                    ModificarListCli(1,2);
                    break;
                }
            case 5:
                {
                    system("cls");
```

Ilustración 51 Menú ejecutivo

```
                    system("cls");
                    ModificarListCli(2,1);
                    break;
                }
            case 2428:
                {
                    system("cls");
                    Direccionescli();
                    break;
                }
            case 0:
                {
                    system("cls");
                    //  printf("Hasta pronto");
                    break;
                }
            default:
                {
                    cout<<"Inserte una opcion valida"<<endl;
                    break;
                }
        }
    }
}


//-------------------------------------------------------------
```

Ilustración 50 Menú ejecutivo

Menús:

Los menús de administrador y ejecutivo son los que contienen las opciones de llamada de funciones necesarias que el usuario solicite. Ambas tienen las mismas opciones, pero manipularan listas diferentes, los administradores trabajaran con las funciones de usuarios y el ejecutivo manipulara los registros de los clientes.

decoracion.h

```
void decomain(){
    SetConsoleTextAttribute(hd, 5);

    for(int a=0;a<120;a++){

        gotoxy(a,29);
        printf("%c",207);

        if(a<30){
            gotoxy(0,a);
            printf("%c",207);

            fflush(stdin);

            gotoxy(119,a);
            printf("%c",207);
        }

        gotoxy(a,0);
        printf("%c",207);
    }

    gotoxy(50,13);
    SetConsoleTextAttribute(hd, 11);
}
```

Ilustración 55 Decoración main

```
void decoprint(){
    HANDLE hd = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hd, 14);

    for(int a=0;a<120;a++){

        gotoxy(a,29);
        printf("%c",177);

        if(a<30){
            gotoxy(0,a);
            printf("%c",177);

            fflush(stdin);

            gotoxy(119,a);
            printf("%c",177);
        }

        gotoxy(a,0);
        printf("%c",177);
    }

    gotoxy(50,13);
    SetConsoleTextAttribute(hd, 11);
}
```

Ilustración 54 Decoración

```
void PrincipioMenu(){
    HANDLE hd = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hd, 11);
    for(int a=0;a<120;a++){ //Bordes

        gotoxy(a,29);
        printf("%c",207);

        if(a<30){
            gotoxy(0,a);
            printf("%c",207);

            fflush(stdin);

            gotoxy(119,a);
            printf("%c",207);
        }

        gotoxy(a,0);
        printf("%c",207);
    }
    SetConsoleTextAttribute(hd, 21);
    gotoxy(50,3);
    cout<<"Bienvenido cliente"<<endl;
    SetConsoleTextAttribute(hd, 13);
    for(int c=5;c<10;c++){  //Letras en y
        gotoxy(35,c);
        printf("%c",186);

        gotoxy(45,c);
        printf("%c",186);

        gotoxy(50,c);
        printf("%c",186);

        gotoxy(53,c);
        printf("%c",186);

        gotoxy(59,c);
        printf("%c",186);

        gotoxy(62,c);
        printf("%c",186);
```

```
        printf("%c",186);

        gotoxy(69,c);
        printf("%c",186);

        gotoxy(75,c);
        printf("%c",186);
    }

    for(int d=36;d<42;d++){ //B
        gotoxy(d,5);
        printf("%c",205);

        gotoxy(d,9);
        printf("%c",205);

        gotoxy(d,7);
        printf("%c",205);

    }

    for(int AA=46;AA<50;AA++){  //A
        gotoxy(AA,5);
        printf("%c",205);

        gotoxy(AA,7);
        printf("%c",205);
    }
    //_____individuales_____
    gotoxy(42,6);       //B
    printf("%c",186);
    gotoxy(42,8);
    printf("%c",186);

    int p=54,q=5;           //N
    while(p!=59){
        gotoxy(p,q);
        printf("%c",187);
        p++;
        q++;
    }

    for(int e=63;e<=66;e++){        //C
```

```
        q++;
    }

    for(int e=63;e<=66;e++){        //C
        gotoxy(e,5);
        printf("%c",205);
        gotoxy(e,9);
        printf("%c",205);
    }

    for(int o=70;o<75;o++){         //O
        gotoxy(o,5);
        printf("%c",205);
        gotoxy(o,9);
        printf("%c",205);
    }
    //_____
    //_____Esquinas Pulidas___
    gotoxy(35,5);       //B
    printf("%c",201);
    gotoxy(35,9);
    printf("%c",200);
    gotoxy(35,7);
    printf("%c",204);

    gotoxy(45,5);           //A
    printf("%c",201);
    gotoxy(50,5);
    printf("%c",187);
    gotoxy(45,7);
    printf("%c",204);
    gotoxy(50,7);
    printf("%c",185);

    gotoxy(53,5);       //N
    printf("%c",201);
    gotoxy(59,9);
    printf("%c",188);

    gotoxy(58,9);
    printf("%c",200);

    gotoxy(62,5);       //C
```

```
177     gotoxy(62,5);       //C
178     printf("%c",201);
179     gotoxy(62,9);
180     printf("%c",200);
181
182     gotoxy(69,9);           //o
183     printf("%c",200);
184     gotoxy(69,5);
185     printf("%c",201);
186     gotoxy(75,9);
187     printf("%c",188);
188     gotoxy(75,5);
189     printf("%c",187);
190
191     //
192     SetConsoleTextAttribute(hd, 23);
193     gotoxy(45,15);
194     printf("Bruno Axel Puente Luna - 177876");
195     gotoxy(45,16);
196     printf("Uriel Montejano Briano - 177771");
197     SetConsoleTextAttribute(hd, 10);
198     getch();
199     system("cls");
200 }
```

```
202 void cuadrologin(){
203     //241
204     SetConsoleTextAttribute(hd, 8);
205     for(int a=10;a<17;a++){
206         gotoxy(45,a);
207         printf("%c",241);
208         gotoxy(78,a);
209         printf("%c",241);
210     }
211     for(int b=45;b<79;b++){
212         gotoxy(b,10);
213         printf("%c",241);
214         gotoxy(b,16);
215         printf("%c",241);
216     }
217 }
```

Funciones decoración:

En esta hoja se trabajo con funciones dedicadas al diseño de los menús que hay en el programa, desde los bordes, posicionamiento del texto y color de las letras. La función gotoxy declarada en la librería, tiene su uso aquí, ya que se necesitan obtener las coordenadas (X, Y) de la pantalla para poder realizar los diseños.

Archivos Generados

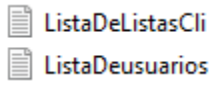

Ilustración 57 Archivos de lista de listas

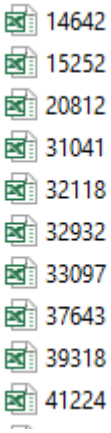

Ilustración 56 Archivos de listas

Los archivos tipo txt so los que contienen los ID de las listas (los archivos tipo xls). Los archivos tipo xls son los que contienen los datos de los usuarios o clientes (dependiendo del registro realizado).

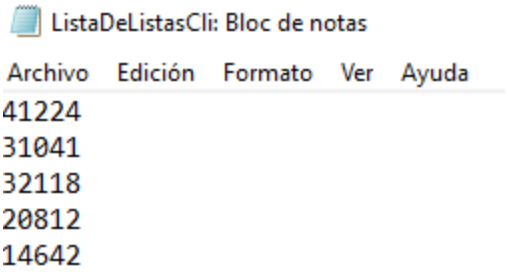

Ilustración 59 Archivo lista de listas de cliente

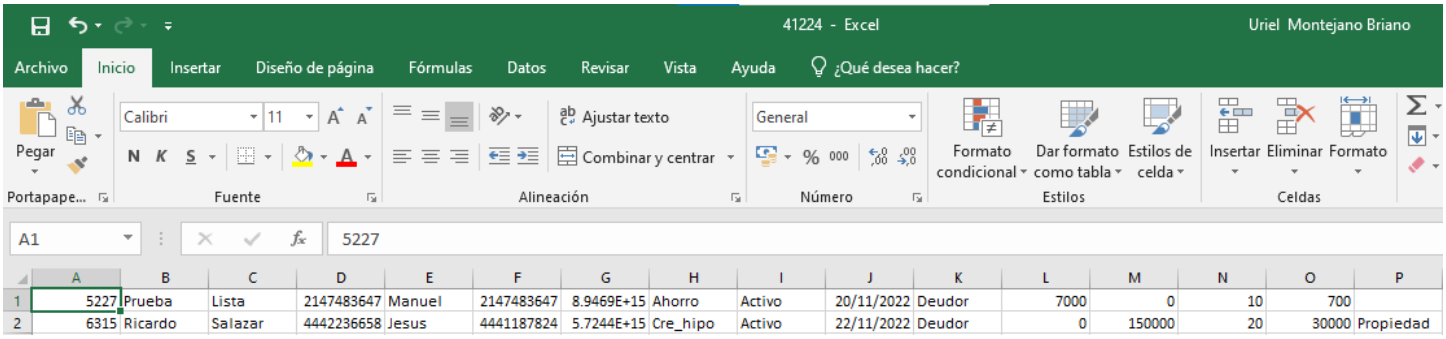

Ilustración 58 Archivo listas de cliente

En estos ejemplos se aprecia cómo es el guardado de datos dependiendo del tipo de estructura, siendo las primeras dos imágenes pertenecientes al usuario tipo ejecutivo, ya que es el encargado de los clientes, y las últimas dos imágenes pertenecen al usuario tipo administrador, ya que es el cargado de los usuarios.

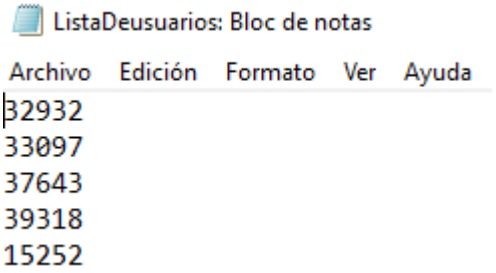

Ilustración 61 Archivo lista de listas usuario

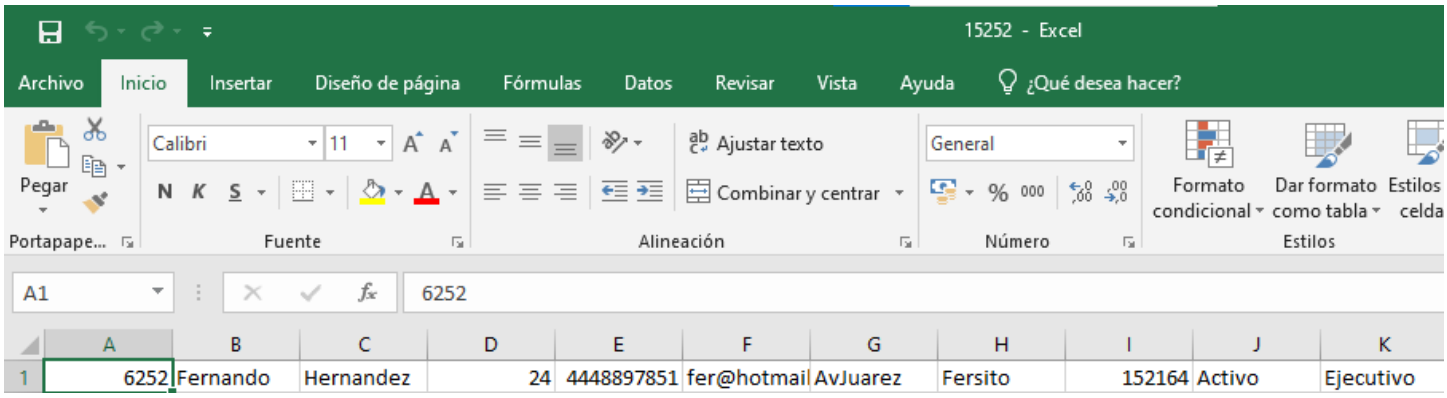

Ilustración 60 Archivo lista usuarios1