

```

26
27 void validaCadena(const char mensaje[], const char cadena[]){
28     bool bandera = true;
29     while(bandera){
30         printf("%s", mensaje);
31         scanf("%[^\\n]", cadena);
32         fflush(stdin);
33         if(strlen(cadena)>29){
34             continue;
35         }else{
36             for(int i=0; i<strlen(cadena); i++){
37                 if(isalpha(cadena[i])|| cadena[i]==' '){
38                     if(i==(strlen(cadena)-1)){
39                         bandera=false;
40                     }
41                 }else{
42                     break;
43                 }
44             }
45         }
46     }
47     return;
48 }

```

Autores:

Montejano Briano Uriel – 177771

Bruno Axel Puente Luna – 177876

Programación II

Prof. Guadalupe Ledesma Ramos

Proyecto Final

23/11/2022

Manual del programador

Contenido

Tabla de ilustraciones	2
INTRODUCCIÓN	4
MARCO TEÓRICO	4
ARCHIVOS EN EL PROYECTO	4
libreria.h	5
Bibliotecas:	5
Declaración de variables:	5
Función gotoxy:	1
Declaración hojas de trabajo:	1
Función login:	2
Main:	3
validacion.h	3
Validaciones:	3
archivos.h y archivosCli.h	4
Listas:	4
Listas de listas:	5
Agregar Final:	7
Agregar nodos en posición:	7
Archivo estructura:	8
Modificar lista:	9
Escritura de listas:	11
Impresión de listas:	14
Número de tarjeta:	15
Menús:	16
Funciones decoración:	17
Archivos Generados	17

Tabla de ilustraciones

Ilustración 1 Hojas del proyecto	4
Ilustración 2 Bibliotecas	5
Ilustración 3 Variables globales	5
Ilustración 5 Estructura usuario y contrato	6
Ilustración 4 Estructura cliente, listas y listas de clientes	6
Ilustración 6 Función gotxy	1
Ilustración 7 Declaración hojas de trabajo	1
Ilustración 8 Funcion login	1
Ilustración 9 Función main	2
Ilustración 10 Función main	2
Ilustración 11 Validación cadena	3
Ilustración 12 Validación flotante y entero	3
Ilustración 13 Lista clientes	4
Ilustración 14 Lista Usuarios	4
Ilustración 15 Lista usuario	4
Ilustración 16 Función listas de listas clientes	5
Ilustración 17 Función listas de listas usuario	5
Ilustración 18 Función contrato	6
Ilustración 19 Función contrato	6
Ilustración 20 Función contrato	6
Ilustración 21 Agregar final usuarios	6
Ilustración 22 Agregar final clientes	6
Ilustración 23 Agregar en posición clientes	7
Ilustración 24 Agregar en posición usuarios	7
Ilustración 25 Escritura de listas usuarios	7
Ilustración 26 Escritura de listas clientes	7
Ilustración 27 Modificar lista cliente	8
Ilustración 28 Modificar lista cliente	8
Ilustración 29 Modificar lista usuario	8
Ilustración 30 Modificar lista usuario	8
Ilustración 31 Modificar lista cliente	9
Ilustración 32 Lectura de archivos cliente	9
Ilustración 33 Lectura de archivos usuario	9
Ilustración 34 Lectura archivos cliente	10
Ilustración 35 Escritura lista de listas usuario	10
Ilustración 36 Escritura lista de listas cliente	10
Ilustración 37 Mostrar lista de clientes	11
Ilustración 38 Mostrar lista de clientes	11
Ilustración 39 Mostrar lista de clientes	12
Ilustración 40 Mostrar lista de clientes	12
Ilustración 41 Mostrar lista de usuarios	12
Ilustración 42 Mostrar lista de clietnes	12
Ilustración 43 Direcciones listas de clientes	13
Ilustración 44 Filtrar clientes	13
Ilustración 45 Filtrar clientes	13
Ilustración 46 Direcciones de clientes	13

Ilustración 47 Direcciones usuario 14

Ilustración 48 Direcciones listas usuarios 14

Ilustración 49 Número de tarjeta aleatorio 14

Ilustración 50 Menú ejecutivo 15

Ilustración 51 Menú ejecutivo 15

Ilustración 52 Menú administrador 15

Ilustración 53 Menú administrador 15

Ilustración 54 Decoración 16

Ilustración 55 Decoración main 16

Ilustración 56 Archivos de listas..... 17

Ilustración 57 Archivos de lista de listas 17

Ilustración 58 Archivo listas de cliente..... 18

Ilustración 59 Archivo lista de listas de cliente 18

Ilustración 60 Archivo lista usuarios1 18

Ilustración 61 Archivo lista de listas usuario 18

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.

```
15 using namespace std;
16
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 definir las 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 nombreccli;
53     string apellidoccli;
54     string telefonoccli;
55     string refecli;
56     string tel_refeccli;
57     int long long tarjetaccli;
58     Contrato deal;
59
60     Cliente *next;
61 }*primeroccli, *ultimoccli;
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 *listaccli;
72     ListasClientes *next;
73 }*primListaccli, *ultListaccli;
74

```

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 gotoxy

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             menujec(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 menujec), 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

```
1 float validaFlotante(const char mensaje[]){
2     int continuar=0;
3     float flotante=0;
4
5     do{
6         printf("%s",mensaje);
7         continuar=scanf("%f", &flotante);
8         fflush(stdin);
9     }while (continuar!=1);
10
11     return flotante;
12 }
13
14 int validaEntero(const char mensaje[]){
15     int continuar=0;
16     int entero=0;
17
18     do{
19         printf("%s",mensaje);
20         continuar=scanf("%d",&entero);
21         fflush(stdin);
22     }while (continuar!=1);
23
24     return entero;
25 }
```

Ilustración 12 Validación flotante y entero

```
27 void validaCadena(const char mensaje[], const char cadena[]){
28     bool bandera = true;
29     while(bandera){
30         printf("%s", mensaje);
31         scanf("%[^\n]",cadena);
32         fflush(stdin);
33         if(strlen(cadena)>29){
34             continue;
35         }else{
36             for(int i=0; i<strlen(cadena); i++){
37                 if(isalpha(cadena[i])|| cadena[i]==' '){
38                     if(i==(strlen(cadena)-1)){
39                         bandera=false;
40                     }
41                 }else{
42                     break;
43                 }
44             }
45         }
46     }
47     return;
48 }
```

Ilustración 11 Validación 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.

```
1 Usuario *registro(){
2     decompain();
3     int opc; hold=1;
4     srand (time(NULL));
5     Usuario *nuevo = new Usuario;
6     nuevo->idUsuario=1000+rand()%(9999-1000)+1;
7     gotoxy(45,5);
8     cout<<"Nombre: ";
9     cin>>nuevo->nombre;
10    gotoxy(45,6);
11    cout<<"Apellido: ";
12    cin>>nuevo->apellido;
13    gotoxy(45,7);
14    nuevo->edad=validaEntero("Edad: ");
15    gotoxy(45,8);
16    cout<<"Telefono: ";
17    cin>>nuevo->telefono;
18    gotoxy(45,9);
19    cout<<"Correo electronico: ";
20    cin>>nuevo->correo;
21    gotoxy(45,10);
22    cout<<"Direccion: ";
23    cin>>nuevo->dir;
24    fflush(stdin);
25    gotoxy(45,11);
26    cout<<"Username: ";
27    cin>>nuevo->username;
28    gotoxy(45,12);
29    cout<<"Password: ";
30    cin>>nuevo->pass;
31    nuevo->estado="Activo";
32    while(hold=1){
33        gotoxy(45,13);
34        cout<<"Perfil: ";gotoxy(45,13);cout<<"1.-Administrador";gotoxy(45,14);cout<<"2.-Ejecutivo";gotoxy(45,15);cout<<"Re ";
35        cin>>opc;
36        gotoxy(48,28);
37        switch(opc){
38            case 1:
39                {
40                    nuevo->perfil=Administrador;
41                    hold=0;
42                    break;
43                }
44            case 2:
45                {
46                    nuevo->perfil=Ejecutivo;
47                    hold=0;
48                    break;
49                }
50            default:
51                {
52                    gotoxy(45,28);
53                    cout<<"Ingrese una opcion valida"<<endl;
54                    break;
55                }
56        }
57    }
58    nuevo->sig=NULL;
59    return nuevo;
60 }
```

Ilustración 15 Lista usuario

```
45 }
46 case 2:
47 {
48     nuevo->perfil=Ejecutivo;
49     hold=0;
50     break;
51 }
52 default:
53 {
54     gotoxy(45,28);
55     cout<<"Ingrese una opcion valida"<<endl;
56     break;
57 }
58 }
59 nuevo->sig=NULL;
60 return nuevo;
61 }
62 }
63 }
```

Ilustración 14 Lista Usuarios

```
129 Cliente *registroCli(){
130     Cliente *nuevo = new Cliente;
131     decompain();
132     srand (time(NULL));
133     fflush(stdin);
134     nuevo->idCliente=1000+rand()%(9999-1000)+1;
135     gotoxy(45,5);
136     cout<<"Nombre: ";
137     cin>>nuevo->nombrecli;
138     gotoxy(45,6);
139     cout<<"Apellido: ";
140     fflush(stdin);
141     cin>>nuevo->apellidocli;
142     gotoxy(45,7);
143     cout<<"Telefono: ";
144     fflush(stdin);
145     cin>>nuevo->telefonocli;
146     gotoxy(45,8);
147     cout<<"Referencia: ";
148     fflush(stdin);
149     cin>>nuevo->refeccli;
150     gotoxy(45,9);
151     cout<<"Telefono de referencia: ";
152     fflush(stdin);
153     cin>>nuevo->tel_refeccli;
154     //-----------------randtarjeta-----
155     generartarjeta();
156     nuevo->tarjetacli=cardnumber;
157     //-----------------
158     makedeal(nuevo);
159     nuevo->next=NULL;
160     return nuevo;
161 }
162 }
```

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         deomain();
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         deomain();
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->listaccli=primerocli;
781     archivoEscrituraCli(primerocli, nuevo->idListasCli);
782
783     if(primListaccli==NULL){
784         primListaccli=nuevo;
785         ultListaccli=nuevo;
786     }else{
787         ultListaccli->next=nuevo;
788         ultListaccli=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.

```

13 void makedeal(Cliente *trato){
14     int opc,hold=1;
15     trato->deal.estadoContra=estado;
16
17     while(hold==1){
18         gotoxy(44,25);
19         cout<<" ";
20         gotoxy(45,10);
21         cout<<"Tipo de cliente:";gotoxy(45,11);cout<<"1.-Ahorrador";gotoxy(45,12);cout<<"2.-Deudor";gotoxy(45,13);cout<<" ";
22         cin>>opc;
23         switch(opc){
24             case 1:
25             {
26                 trato->deal.tipClien=Ahorrador;
27                 hold=0;
28                 break;
29             }
30             case 2:
31             {
32                 trato->deal.tipClien=Deudor;
33                 hold=0;
34                 break;
35             }
36             default:
37             {
38                 gotoxy(45,25);
39                 cout<<"Inserte una opcion valida";
40                 break;
41             }
42         }
43     }
44
45     opc=0;
46     hold=1;
47     gotoxy(45,14);
48     cout<<"Fecha: ";
49     cin>>trato->deal.fechaA;
50     do{
51         gotoxy(44,28);
52         cout<<" ";
53         fflush(stdin);
54         gotoxy(45,15);
55         cout<<"Tipo de tarjeta del cliente: ";
56         gotoxy(45,16);
57         cout<<"1.-Ahorro";gotoxy(45,17);cout<<"2.-Credito";gotoxy(45,18);cout<<"3.-Debito";gotoxy(45,19);cout<<"4.-Credito";gotoxy(45,20);cout<<" ";

```

Ilustración 20 Función contrato

```

57     cout<<"1.-Ahorro";gotoxy(45,17);cout<<"2.-Credito";gotoxy(45,18);cout<<"3.-Debito";gotoxy(45,19);cout<<"4.-Credito";gotoxy(45,20);cout<<" ";
58     cin>>opc;
59     switch(opc){
60         case 1:
61         {
62             trato->deal.prestamo=0;
63             trato->deal.tipotarcli=Ahorro;
64             gotoxy(45,21);
65             trato->deal.CanAhorro=validaFlotante("Cantidad a ahorrar: ");
66             trato->deal.interes=10;
67             gotoxy(45,22);
68             cout<<"Interes sobre ahorro es del 10% por mes";
69             trato->deal.anio=0.1*trato->deal.CanAhorro;
70             gotoxy(45,23);
71             cout<<"Cantidad acumulada dentro de un año: $"<<trato->deal.anio;
72             hold=0;
73             break;
74         }
75         case 2:
76         {
77             trato->deal.CanAhorro=0;
78             trato->deal.tipotarcli=Credito;
79             fflush(stdin);
80             gotoxy(45,21);
81             trato->deal.prestamo=validaFlotante("Cantidad de prestamo: ");
82             trato->deal.interes=15;
83             gotoxy(45,22);
84             cout<<"Interes de credito es del 15% por mes";
85             trato->deal.anio=0.15*trato->deal.prestamo;
86             gotoxy(45,23);
87             cout<<"Cantidad de cobro anual por uso de la tarjeta: $"<<trato->deal.anio;
88             hold=0;
89             break;
90         }
91         case 3:
92         {
93             trato->deal.tipotarcli=Debito;
94             hold=0;
95             break;
96         }
97         case 4:
98         {
99             trato->deal.CanAhorro=0;
100            trato->deal.tipotarcli=Creditohipotecario;
101            gotoxy(45,21);

```

Ilustración 19 Función contrato

```

101            gotoxy(45,21);
102            cout<<"Que bien va a hipotecar el cliente? ";
103            cin>>trato->deal.hipotecaref;
104            gotoxy(45,22);
105            trato->deal.prestamo=validaEntero("Valor del avaluo dado: ");
106            trato->deal.interes=20;
107            gotoxy(45,23);
108            cout<<"Interes de la hipoteca es del 20% por mes";
109            trato->deal.anio=0.20*trato->deal.prestamo;
110            gotoxy(45,24);
111            cout<<"Cantidad de cobro anual por la hipoteca: $"<<trato->deal.anio;
112            hold=0;
113            break;
114        }
115        default:
116        {
117            gotoxy(45,28);
118            cout<<"Ingrese una opcion valida"<<endl;
119            break;
120        }
121    }
122 }while(hold==1);
123 getch();
124 system("cls");
125 deomain();
126 fflush(stdin);
127 }
128

```

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.

```

164 void agregarFinalCli(){
165     Cliente *nuevo = registroCli();
166     if(primerocli==NULL){
167         primerocli=nuevo;
168         ultimocli=nuevo;
169     }
170     else{
171         ultimocli->next=nuevo;
172         ultimocli=nuevo;
173     }
174 }
175

```

Ilustración 22 Agregar final clientes

```

66 void agregarFinal(){
67     Usuario *nuevo = registro();
68     if(primerocli==NULL){
69         primero=nuevo;
70         ultimo=nuevo;
71     }
72     else{
73         ultimo->sig=nuevo;
74         ultimo=nuevo;
75     }
76 }

```

Ilustración 21 Agregar final usuarios

Agregar Final:

Las funciones agregarFinal 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.

```
70 void agregarAlistita(Usuario *primero){
71     system("cls");
72     deomain();
73     struct Usuario *auxiliar=primero;
74     bool bandera=true;
75     int pos = 0;
76     int i = 1;
77     gotoxy(45,5);
78     pos = validaEntero("Posición: ");
79     if(pos==1){
80         Usuario *nuevo = registro();
81         nuevo->sig=primero;
82         primero=nuevo;
83     }
84     else{
85         while(auxiliar!=NULL){
86             if(i == (pos-1)){
87                 Usuario *nuevo = registro();
88                 nuevo->sig = auxiliar->sig;
89                 auxiliar->sig = nuevo;
90                 bandera=false;
91                 break;
92             }
93             else{
94                 auxiliar = auxiliar->sig;
95                 i++;
96             }
97         }
98         if(bandera){
99             gotoxy(30,28);
100             cout<<"La posición NO existe, el nodo se agrega al final de la lista"<<endl;
101             agregarFinal();
102         }
103     }
104     getch();
105 }
```

Ilustración 24 Agregar en posición usuarios

```
176 void agregarAlistitaCli(Cliente *primerocli){
177     system("cls");
178     deomain();
179     struct Cliente *auxiliar=primerocli;
180     bool bandera=true;
181     int pos = 0;
182     int i = 1;
183     gotoxy(45,5);
184     pos = validaEntero("Posición: ");
185     system("cls");
186     if(pos==1){
187         Cliente *nuevo = registroCli();
188         nuevo->next=primerocli;
189         primerocli=nuevo;
190     }
191     else{
192         while(auxiliar!=NULL){
193             if(i == (pos-1)){
194                 Cliente *nuevo = registroCli();
195                 nuevo->next = auxiliar->next;
196                 auxiliar->next = nuevo;
197                 bandera=false;
198                 break;
199             }
200             else{
201                 auxiliar = auxiliar->next;
202                 i++;
203             }
204         }
205         if(bandera){
206             gotoxy(30,28);
207             cout<<"La posición NO existe, el nodo se agrega al final de la lista"<<endl;
208             agregarFinalCli();
209         }
210     }
211     getch();
212 }
```

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.

```
212 void archivoEscrituraCli(Cliente *primerocli,int id){
213     Cliente *aux=primerocli;
214     char arch[20];
215     itoa(id,arch,10);
216     strcat(arch,".xls");
217     ofstream archivo(arch);
218
219     while(aux!=NULL){
220         archivo<<aux->idCli<<"\t";
221         archivo<<aux->nombrecli<<"\t";
222         archivo<<aux->apellidocli<<"\t";
223         archivo<<aux->telefonocli<<"\t";
224         archivo<<aux->refeccli<<"\t";
225         archivo<<aux->tel_refeccli<<"\t";
226         archivo<<aux->tarjetacli<<"\t";
227
228         archivo<<aux->deal.tipotarcli<<"\t";
229         archivo<<aux->deal.estadoContra<<"\t";
230         archivo<<aux->deal.fechaA<<"\t";
231         archivo<<aux->deal.tipClien<<"\t";
232         archivo<<aux->deal.CanAhorro<<"\t";
233         archivo<<aux->deal.prestamo<<"\t";
234         archivo<<aux->deal.interes<<"\t";
235         archivo<<aux->deal.anio<<"\t";
236         archivo<<aux->deal.hipotecaref<<endl;
237
238         ultimocli=aux;
239         aux=aux->next;
240     }
241     archivo.close();
242 }
243 }
```

Ilustración 26 Escritura de listas clientes

```
116 void archivoEscritura(Usuario *primero,int id){
117     Usuario *aux=primero;
118     char arch[20];
119     itoa(id,arch,10);
120     strcat(arch,".xls");
121     ofstream archivo(arch);
122
123     while(aux!=NULL){
124         archivo<<aux->idUsuario<<"\t";
125         archivo<<aux->nombre<<"\t";
126         archivo<<aux->apellido<<"\t";
127         archivo<<aux->edad<<"\t";
128         archivo<<aux->telefono<<"\t";
129         archivo<<aux->correo<<"\t";
130         archivo<<aux->dir<<"\t";
131         archivo<<aux->username<<"\t";
132         archivo<<aux->pass<<"\t";
133         archivo<<aux->estado<<"\t";
134         archivo<<aux->perfil<<endl;
135         ultimo=aux;
136         aux=aux->sig;
137     }
138     archivo.close();
139 }
```

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.

```

141 void modificalistita(Usuario *primero, int estado, int aa){
142     struct Usuario *auxiliar=primero;
143     int folmod, op, hold=1, mod=1, bb=aa+3, opc;
144
145     gotoxy(45, aa+3);
146     cout<<"ID's de la lista seleccionada";
147     while(primero!=NULL){
148         bb+=1;
149         gotoxy(45, bb);
150         cout<<primero->idUsuario<<" ---->"<<primero->nombre;
151         primero=primero->sig;
152     }
153
154     while(mod==1){
155         gotoxy(45, bb+1);
156         folmod=validaEntero("Selecciona el ID del Usuario a modificar: ");
157         gotoxy(3, 28);
158         cout<<" ";
159         bool bandera = true;
160         while(bandera){
161             primero=auxiliar;
162             while(primero!=NULL){
163                 if(folmod != primero->idUsuario){
164                     primero=primero->sig;
165                     bandera=false;
166                 }else{
167                     bandera = true;
168                     break;
169                 }
170             }
171             if(bandera==false){
172                 gotoxy(3, 28);
173                 cout<<"El ID no esta registrado"<<endl;
174             }else{
175                 bandera=false;
176                 mod=2;
177             }
178         }
179     }
180     system("cls");
181     deomain();
182     hold=1;
183     while(auxiliar!=NULL){
184         if(auxiliar->idUsuario==folmod){

```

Ilustración 30 Modificar lista usuario

```

184         if(auxiliar->idUsuario==folmod){
185             if(estado==1){
186                 gotoxy(45, 3);
187                 cout<<"Nombre: ";
188                 cin>>auxiliar->nombre;
189                 gotoxy(45, 4);
190                 cout<<"Apellido: ";
191                 cin>>auxiliar->apellido;
192                 gotoxy(45, 5);
193                 auxiliar->edad=validaEntero("Edad: ");
194                 gotoxy(45, 6);
195                 cout<<"Telefono: ";
196                 cin>>auxiliar->telefono;
197                 gotoxy(45, 7);
198                 cout<<"Correo electronico: ";
199                 cin>>auxiliar->correo;
200                 gotoxy(45, 8);
201                 cout<<"Direccion: ";
202                 cin>>auxiliar->dir;
203                 gotoxy(45, 9);
204                 cout<<"Username: ";
205                 cin>>auxiliar->username;
206                 gotoxy(45, 10);
207                 cout<<"Password: ";
208                 cin>>auxiliar->pass;
209             }
210             while(hold==1){
211                 gotoxy(45, 13);
212                 cout<<"Perfil: "; gotoxy(45, 14); cout<<"1.-Administrador"; gotoxy(45, 15); cout<<"2.-Ejecutivo"; cout<<"R=";
213                 cin>>opc;
214                 gotoxy(45, 28);
215                 cout<<" ";
216                 switch(opc){
217                     case 1:
218                         {
219                             auxiliar->xperfil=Administrador;
220                             hold=0;
221                             break;
222                         }
223                     case 2:
224                         {
225                             auxiliar->xperfil=Ejecutivo;
226                             hold=0;
227                             break;
228                         }
229                 }
230             }
231         }

```

Ilustración 29 Modificar lista usuario

```

245 void modificalistitaCli(Cliente *primerocli, int estado, int aa){
246     struct Cliente *auxiliar=primerocli;
247     int folmod, op, hold=1, mod=1, bb=aa+3;
248     gotoxy(45, aa+3);
249     cout<<"ID's de la lista seleccionada"<<endl;
250     while(primeroCli!=NULL){
251         bb+=1;
252         gotoxy(45, bb);
253         cout<<primeroCli->idCliente<<" ---->"<<primeroCli->nombreCli<<endl;
254         primeroCli=primeroCli->next;
255     }
256
257     while(mod==1){
258         gotoxy(45, bb+1);
259         folmod=validaEntero("Selecciona el ID del Usuario a modificar: ");
260         gotoxy(45, 28);
261         cout<<" ";
262         bool bandera = true;
263         while(bandera){
264             primeroCli=auxiliar;
265             while(primeroCli!=NULL){
266                 if(folmod != primeroCli->idCliente){
267                     primeroCli=primeroCli->next;
268                     bandera=false;
269                 }else{
270                     bandera = true;
271                     break;
272                 }
273             }
274             if(bandera==false){
275                 gotoxy(45, 28);
276                 cout<<"El ID no esta registrado"<<endl;
277             }else{
278                 bandera=false;
279                 mod=2;
280             }
281         }
282     }
283     system("cls");
284     deomain();
285     primeroCli=auxiliar;
286     int opc;
287     hold=1;

```

Ilustración 28 Modificar lista cliente

```

289     while(auxiliar!=NULL){
290         if(auxiliar->idCliente==folmod){
291             if(estado==1){
292                 gotoxy(45, 3);
293                 auxiliar->telefonocli=validaEntero("Telefono Cliente: ");
294                 gotoxy(45, 4);
295                 cout<<"Referencia: ";
296                 cin>>auxiliar->refeccli;
297                 gotoxy(45, 5);
298                 auxiliar->tel_refeccli=validaEntero("Telefono Referencia: ");
299                 gotoxy(45, 6);
300                 while(hold==1){
301                     gotoxy(45, 7);
302                     cout<<"Tipo de cliente:"; gotoxy(45, 8); cout<<"1.-Ahorrador"; gotoxy(45, 9); cout<<"2.-Ejecutivo"; cout<<"R=";
303                     cin>>opc;
304                     switch(opc){
305                         case 1:
306                             {
307                                 auxiliar->deal.tipClien=Ahorrador;
308                                 hold=0;
309                                 break;
310                             }
311                         case 2:
312                             {
313                                 auxiliar->deal.tipClien=DEudor;
314                                 hold=0;
315                                 break;
316                             }
317                         default:
318                             {
319                                 break;
320                             }
321                     }
322                 }
323             }
324             if(estado==2){
325                 gotoxy(45, 13);
326                 cout<<"Estado del usuario: ";
327                 cin>>auxiliar->estado;
328                 gotoxy(45, 28);
329                 cout<<"Ingrese una opcion valida"<<endl;
330                 break;
331             }
332         }
333     }

```

Ilustración 27 Modificar lista cliente

```

240     gotoxy(45, 13);
241     cout<<"Estado del usuario: ";
242     cin>>auxiliar->estado;
243     gotoxy(45, 28);
244     cout<<"Ingrese una opcion valida"<<endl;
245     break;
246 }
247 }

```



```

333         hold=0;
334         break;
335     }
336     case 2:
337     {
338         auxiliar->deal.estadoContra=Inactivo;
339         hold=0;
340         break;
341     }
342     case 3:
343     {
344         auxiliar->deal.estadoContra=En_Deuda;
345         hold=0;
346         break;
347     }
348     case 4:
349     {
350         auxiliar->deal.estadoContra=Cancelado;
351         hold=0;
352         break;
353     }
354     case 5:
355     {
356         auxiliar->deal.estadoContra=Sin_Deuda;
357         hold=0;
358         break;
359     }
360     default:
361     {
362         break;
363     }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

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.

```

311 Usuario *archivoLectura(int id){
312     Usuario *primero=NULL;
313     Usuario *ultimo=NULL;
314     Usuario *nuevo=NULL;
315
316     char archivo[20];
317     itoa(id,archivo,10);
318     strcat(archivo,".xls");
319
320     ifstream arch(archivo);
321     string linea, c;
322
323     cout<<id<<endl;
324     if(arch.fail()) cerr<<"Error al abrir el archivo"<<endl;
325     else{
326         while(getline(arch,linea)){
327             nuevo = new Usuario;
328             stringstream lee(linea);
329
330             lee>>nuevo->idUsuario;
331             getline(lee,c,'\t');
332             getline(lee,nuevo->nombre,'\t');
333             getline(lee,nuevo->apellido,'\t');
334             lee>>nuevo->edad;
335             getline(lee,c,'\t');
336             lee>>nuevo->telefono;
337             getline(lee,c,'\t');
338             getline(lee,nuevo->correo,'\t');
339             getline(lee,nuevo->xdir,'\t');
340             getline(lee,nuevo->username,'\t');
341             getline(lee,nuevo->pass,'\t');
342             getline(lee,nuevo->estado,'\t');
343             getline(lee,nuevo->perfil,'\n');
344
345             nuevo->sig=NULL;
346             if(primero==NULL){
347                 primero = nuevo;
348                 ultimo=nuevo;
349             }else{
350                 ultimo->sig=nuevo;
351                 ultimo=nuevo;
352             }
353         }
354     }
355     arch.close();

```

Ilustración 33 Lectura de archivos usuario

```

432 Cliente *archivoLecturaCli(int id){
433     Cliente *primerocli=NULL;
434     Cliente *ultimocli=NULL;
435     Cliente *nuevo=NULL;
436
437     char archivo[20];
438     itoa(id,archivo,10);
439     strcat(archivo,".xls");
440
441     ifstream arch(archivo);
442     string linea, c;
443
444     cout<<id<<endl;
445     if(arch.fail()) cerr<<"Error al abrir el archivo"<<endl;
446     else{
447         while(getline(arch,linea)){
448             nuevo = new Cliente;
449             stringstream lee(linea);
450
451             lee>>nuevo->idCliente;
452             getline(lee,c,'\t');
453             getline(lee,nuevo->nombrecli,'\t');
454             getline(lee,nuevo->apellidocli,'\t');
455             lee>>nuevo->telefonocli;
456             getline(lee,c,'\t');
457             getline(lee,nuevo->refeccli,'\t');
458             lee>>nuevo->tel_refeccli;
459             getline(lee,c,'\t');
460             lee>>nuevo->tarjetacli;
461             getline(lee,c,'\t');
462
463             getline(lee,nuevo->deal.tipotarcli,'\t');
464             getline(lee,nuevo->deal.estadoContra,'\t');
465             getline(lee,nuevo->deal.fechaA,'\t');
466             getline(lee,nuevo->deal.tipClien,'\t');
467
468             lee>>nuevo->deal.CanAhorro;
469             getline(lee,c,'\t');
470             lee>>nuevo->deal.prestamo;
471             getline(lee,c,'\t');
472             lee>>nuevo->deal.interes;
473             getline(lee,c,'\t');
474             lee>>nuevo->deal.anio;
475             getline(lee,c,'\t');
476             lee>>nuevo->deal.hipotecaref,'\n');
477

```

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. Estas 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 escrituraListacli.txt(){
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->nombreccli<<setw(15);
565         cout<<auxiliar->apellidoccli<<setw(20);
566         cout<<auxiliar->telefonoccli<<setw(15);
567         cout<<auxiliar->refeccli<<setw(15);
568         cout<<auxiliar->tel_refeccli<<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->nombreccli<<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

```

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->refeccli<<setw(15);
568         cout<<auxiliar->tel_refeccli<<setw(21);
569         cout<<auxiliar->tarjetacli<<setw(15);
570         cout<<auxiliar->deal.estadoContra<<endl;
571
572         auxiliar=auxiliar->next;
573     }
574     cout<<" "<<endl;
575 }

```

Ilustración 42 Mostrar lista de clietnes

```

405 void mostrar_lista(Usuario *primero){
406     struct Usuario *auxiliar=primero;
407     cout<<" \n"<<endl;
408     cout<<"Mostrando la lista completa"<<endl;
409     SetConsoleTextAttribute(hd, 12);
410     cout<<"--ID--"<<setw(11);
411     cout<<"Nombre"<<setw(15);
412     cout<<"Apellido"<<setw(11);
413     cout<<"Edad"<<setw(15);
414     cout<<"Telefono"<<setw(20);
415     cout<<"Correo"<<setw(20);
416     cout<<"Direccion"<<setw(20);
417     cout<<"Username"<<setw(15);
418     cout<<"Password"<<setw(15);
419     cout<<"Estado"<<setw(13);
420     cout<<"Perfil"<<endl;
421
422     SetConsoleTextAttribute(hd, 9);
423     while(auxiliar!=NULL){
424         cout<<auxiliar->idUsuario<<setw(11);
425         cout<<auxiliar->nombre<<setw(15);
426         cout<<auxiliar->apellido<<setw(11);
427         cout<<auxiliar->edad<<setw(15);
428         cout<<auxiliar->telefono<<setw(20);
429         cout<<auxiliar->correo<<setw(20);
430         cout<<auxiliar->dir<<setw(20);
431         cout<<auxiliar->username<<setw(15);
432         cout<<auxiliar->pass<<setw(15);
433         cout<<auxiliar->estado<<setw(13);
434         cout<<auxiliar->perfil<<endl;
435         auxiliar=auxiliar->sig;
436     }
437
438     //Mostrar LISTOTA
439     void mostrar_listas(){
440         system("cls");
441         struct Listas *aux=primLista;
442
443         aux=primLista;
444         while(aux!=NULL){
445             SetConsoleTextAttribute(hd, 10);
446             cout<<endl<<"ID de la lista: "<<aux->idListas;
447             mostrar_lista(aux->lista);
448             aux=aux->sig;
449         }
450         getch();
451     }
452 }

```

Ilustración 41 Mostrar lista de usuarios

```

688 void mostrar_listasCli(){
689     system("cls");
690     struct ListasCli *aux=primListasCli;
691     int opc,hold=1,aa=0;
692
693     aux=primListasCli;
694     while(aux!=NULL){
695         SetConsoleTextAttribute(hd, 10);
696         cout<<endl<<"ID de la lista: "<<aux->idListasCli;
697         mostrar_listaCli(aux->listascli);
698         aa++;
699         aux=aux->next;
700     }
701
702     do{
703         gotoxy(40,aa+1);
704         SetConsoleTextAttribute(hd, 14);
705         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
706         cin>>opc;
707         aa++;
708         system("cls");
709         switch(opc){
710             case 1:
711                 aux=primListasCli;
712                 while(aux!=NULL){
713                     SetConsoleTextAttribute(hd, 10);
714                     cout<<endl<<"ID lista: "<<aux->idListasCli;
715                     checarcontrato(aux->listascli);
716                     aa++;
717                     aux=aux->next;
718                 }
719                 break;
720             case 2:
721                 aux=primListasCli;
722                 while(aux!=NULL){
723                     SetConsoleTextAttribute(hd, 10);
724                     cout<<endl<<"ID lista: "<<aux->idListasCli;
725                     filtrar(aux->listascli,1);
726                     aa++;
727                     aux=aux->next;
728                 }
729                 break;
730             case 3:
731                 aux=primListasCli;
732                 while(aux!=NULL){
733                     SetConsoleTextAttribute(hd, 10);
734                     cout<<endl<<"ID lista: "<<aux->idListasCli;
735                     filtrar(aux->listascli,2);
736                     aa++;
737                     aux=aux->next;
738                 }
739                 break;
740             case 0:
741                 hold=0;
742                 break;
743             default:
744                 cout<<"Ingrese una opcion valida"<<endl;
745                 break;
746         }
747     }while(hold==1);
748 }

```

Ilustración 40 Mostrar lista de clientes

```

712 {
713     aux=primListasCli;
714     while(aux!=NULL){
715         SetConsoleTextAttribute(hd, 10);
716         cout<<endl<<"ID lista: "<<aux->idListasCli;
717         checarcontrato(aux->listascli);
718         aa++;
719         aux=aux->next;
720     }
721     break;
722
723     case 2:
724     {
725         aux=primListasCli;
726         while(aux!=NULL){
727             SetConsoleTextAttribute(hd, 10);
728             cout<<endl<<"ID lista: "<<aux->idListasCli;
729             filtrar(aux->listascli,1);
730             aa++;
731             aux=aux->next;
732         }
733         break;
734
735     case 3:
736     {
737         aux=primListasCli;
738         while(aux!=NULL){
739             SetConsoleTextAttribute(hd, 10);
740             cout<<endl<<"ID lista: "<<aux->idListasCli;
741             filtrar(aux->listascli,2);
742             aa++;
743             aux=aux->next;
744         }
745         break;
746
747     case 0:
748     {
749         hold=0;
750         break;
751
752     default:
753     {
754         cout<<"Ingrese una opcion valida"<<endl;
755         break;
756     }
757
758     }while(hold==1);
759 }

```

Ilustración 39 Mostrar lista de clientes

```

619 void filtrar(Cliente *primerocli, int DEAH){
620     struct Cliente *auxiliar=primerocli;
621     cout<<" "<<endl;
622     SetConsoleTextAttribute(hd, 12);
623     cout<<"--ID--"<<setw(10);
624     cout<<"Nombre"<<setw(15);
625     cout<<"Fecha"<<setw(20);
626     cout<<"No. de tarjeta"<<setw(20);
627     cout<<"Tip Tarjeta"<<setw(12);
628     cout<<"Prestamo"<<setw(15);
629     cout<<"Est Contrato"<<setw(15);
630     cout<<"Tip Cliente"<<setw(10);
631     cout<<"Interes"<<setw(12);
632     cout<<"IGA"<<setw(10); //Ingreso generado en un año
633     cout<<"DGA"<<setw(10); //Deuda generada en un año
634     cout<<"Hipoteca"<<endl;
635     if(DEAH==2){
636         if(auxiliar->deal.tipClien==deudOR or auxiliar->deal.tipClien==DEudor){
637             SetConsoleTextAttribute(hd, 9);
638             while(auxiliar!=NULL){
639                 cout<<auxiliar->idClien<<setw(10);
640                 cout<<auxiliar->nombreccli<<setw(15);
641                 cout<<auxiliar->deal.fechaA<<setw(20);
642                 cout<<auxiliar->tarjetacli<<setw(20);
643                 cout<<auxiliar->deal.tipotarcli<<setw(12);
644                 cout<<auxiliar->deal.prestamo<<setw(15);
645                 cout<<auxiliar->deal.estadoContra<<setw(15);
646                 cout<<auxiliar->deal.tipClien<<setw(10);
647                 cout<<auxiliar->deal.interes<<"%"<<setw(12);
648                 if((auxiliar->deal.tipClien==DEudor)|| (auxiliar->deal.tipClien==deudOR)){
649                     cout<<" "<<setw(10);
650                     cout<<auxiliar->deal.anio<<setw(10);
651                 }else{
652                     cout<<auxiliar->deal.anio<<setw(10);
653                     cout<<" "<<setw(10);
654                 }
655                 cout<<auxiliar->deal.hipotecaref<<endl;
656                 auxiliar=auxiliar->next;
657             }
658             cout<<" "<<endl;
659         }
660     }
661     if(DEAH==1){
662         if(auxiliar->deal.tipClien==Ahorrador or auxiliar->deal.tipClien==ahorrador){
663             SetConsoleTextAttribute(hd, 9);
664             while(auxiliar!=NULL){
665                 cout<<auxiliar->idClien<<setw(10);
666                 cout<<auxiliar->nombreccli<<setw(15);

```

Ilustración 45 Filtrar clientes

```

793 void dircli(Cliente *primerocli){
794     struct Cliente *auxiliar=primerocli;
795     cout<<" "<<endl;
796     SetConsoleTextAttribute(hd, 12);
797     cout<<"Ubicacion"<<setw(10);
798     cout<<"--ID--"<<setw(10);
799     cout<<"Nombre"<<setw(15);
800     cout<<"Apellido"<<setw(20);
801     cout<<"Telefono"<<setw(15);
802     cout<<"Referencia"<<setw(15);
803     cout<<"Tel Ref"<<setw(21);
804     cout<<"No. de tarjeta"<<setw(15);
805     cout<<"Est Contrat"<<setw(15);
806     cout<<"siguiente"<<endl;
807
808     SetConsoleTextAttribute(hd, 9);
809     while(auxiliar!=NULL){
810         cout<<auxiliar<<setw(10);
811         cout<<auxiliar->idClien<<setw(11);
812         cout<<auxiliar->nombreccli<<setw(15);
813         cout<<auxiliar->apellidoccli<<setw(20);
814         cout<<auxiliar->telefonoccli<<setw(15);
815         cout<<auxiliar->refeccli<<setw(15);
816         cout<<auxiliar->tel_refeccli<<setw(21);
817         cout<<auxiliar->tarjetacli<<setw(15);
818         cout<<auxiliar->deal.estadoContra<<setw(15);
819         cout<<auxiliar->next<<endl;
820         auxiliar=auxiliar->next;
821     }
822
823     cout<<endl<<"Primer nodo"<<primerocli;
824     cout<<endl<<"Ultimo nodo"<<ultimocli;
825     cout<<" "<<endl;
826 }

```

Ilustración 46 Direcciones de clientes

```

661 if(DEAH==1){
662     if(auxiliar->deal.tipClien==Ahorrador or auxiliar->deal.tipClien==ahorrador){
663         SetConsoleTextAttribute(hd, 9);
664         while(auxiliar!=NULL){
665             cout<<auxiliar->idClien<<setw(10);
666             cout<<auxiliar->nombreccli<<setw(15);
667             cout<<auxiliar->deal.fechaA<<setw(20);
668             cout<<auxiliar->tarjetacli<<setw(20);
669             cout<<auxiliar->deal.tipotarcli<<setw(12);
670             cout<<auxiliar->deal.prestamo<<setw(15);
671             cout<<auxiliar->deal.estadoContra<<setw(15);
672             cout<<auxiliar->deal.tipClien<<setw(10);
673             cout<<auxiliar->deal.interes<<"%"<<setw(12);
674             if((auxiliar->deal.tipClien==DEudor)|| (auxiliar->deal.tipClien==deudOR)){
675                 cout<<" "<<setw(10);
676                 cout<<auxiliar->deal.anio<<setw(10);
677             }else{
678                 cout<<auxiliar->deal.anio<<setw(10);
679                 cout<<" "<<setw(10);
680             }
681             cout<<auxiliar->deal.hipotecaref<<endl;
682             auxiliar=auxiliar->next;
683         }
684         cout<<" "<<endl;
685     }
686 }
687 }

```

Ilustración 44 Filtrar clientes

```

828 void Direccionescli(){
829     system("cls");
830     struct ListasClientes *aux=primListaccli;
831     cout<<"Ubicacion Lista"<<setw(20);
832     cout<<"Lista(Primer nodo)"<<setw(20);
833     cout<<"Siguiente"<<endl;
834
835     while(aux!=NULL){
836         cout<<aux<<setw(20);
837         cout<<aux->listaccli<<setw(20);
838         cout<<aux->next<<endl;
839         aux=aux->next;
840     }
841     cout<<endl<<"Primer LISTA"<<primListaccli;
842     cout<<endl<<"Ultima LISTA"<<ultListaccli;
843
844     aux=primListaccli;
845     while(aux!=NULL){
846         SetConsoleTextAttribute(hd, 10);
847         cout<<endl<<"ID de la lista: "<<aux->idListascli;
848         dircli(aux->listaccli);
849         aux=aux->next;
850     }
851     getch();
852     system("cls");
853 }

```

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.

```
542 void menuadmin(string name){
543     system("cls");
544     int op=1;
545     deomain();
546     gotoxy(40,10);
547     cout<<"--->Bienvenido Aministrador "<<name<<"---<<endl;
548     while(op!=0){
549         deomain();
550         gotoxy(40,12);
551         cout<<"1.- Consultar Listas";gotoxy(40,13);cout<<"2.-Agregar Nueva lis
552         gotoxy(40,18);
553         op = validaEntero("Que desea hacer: ");
554         switch(op){
555             case 1:
556             {
557                 system("cls");
558                 mostrar_listas();
559                 system("cls");
560                 break;
561             }
562             case 2:
563             {
564                 system("cls");
565                 deomain();
566                 agregarLista();
567                 escrituralistatxt();
568                 break;
569             }
570             case 3:
571             {
572                 system("cls");
573                 ModificarList(1,1);
574                 break;
575             }
576             case 4:
577             {
578                 system("cls");
579                 ModificarList(1,2);
580                 break;
581             }
582         }
583     }
584 }
```

Ilustración 53 Menú administrador

```
610 void menujec(string name){
611     system("cls");
612     int op=1;
613     deomain();
614     gotoxy(40,10);
615     cout<<"--->Bienvenido Ejecutivo "<<name<<"---<<endl;
616     while(op!=0){
617         deomain();
618         gotoxy(40,12);
619         cout<<"1.- Consultar Listas de clientes";gotoxy(40,13);cout<<
620         gotoxy(40,18);
621         op = validaEntero("Que desea hacer: ");
622         switch(op){
623             case 1:
624             {
625                 system("cls");
626                 mostrar_listasCli();
627                 system("cls");
628                 break;
629             }
630             case 2:
631             {
632                 system("cls");
633                 agregarListaCli();
634                 escrituralistaClixtxt();
635                 break;
636             }
637             case 3:
638             {
639                 system("cls");
640                 ModificarListCli(1,1);
641                 break;
642             }
643             case 4:
644             {
645                 system("cls");
646                 ModificarListCli(1,2);
647                 break;
648             }
649             case 5:
650             {
651                 system("cls");
652                 break;
653             }
654         }
655     }
656 }
```

Ilustración 51 Menú ejecutivo

```
580     break;
581 }
582 case 5:
583 {
584     system("cls");
585     ModificarList(2,1);
586     break;
587 }
588 case 0:
589 {
590     system("cls");
591     break;
592 }
593 case 2428:
594 {
595     system("cls");
596     direcciones();
597     break;
598 }
599 default:
600 {
601     cout<<"Ingrese una opcion valida"<<endl;
602 }
603 }
604 }
605 }
606 }
```

Ilustración 52 Menú administrador

```
651     system("cls");
652     ModificarListCli(2,1);
653     break;
654 }
655 case 2428:
656 {
657     system("cls");
658     Direccionescli();
659     break;
660 }
661 case 0:
662 {
663     system("cls");
664     // printf("Hasta pronto");
665     break;
666 }
667 default:
668 {
669     cout<<"Inserte una opcion valida"<<endl;
670     break;
671 }
672 }
673 }
674 }
675 }
676 }
677 //-----
```

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

```
1 void decomain(){
2     SetConsoleTextAttribute(hd, 5);
3
4     for(int a=0;a<120;a++){
5
6         gotoxy(a,29);
7         printf("%c",207);
8
9         if(a<30){
10             gotoxy(0,a);
11             printf("%c",207);
12
13             fflush(stdin);
14
15             gotoxy(119,a);
16             printf("%c",207);
17         }
18
19         gotoxy(a,0);
20         printf("%c",207);
21     }
22
23     gotoxy(50,13);
24     SetConsoleTextAttribute(hd, 11);
25 }
```

Ilustración 55 Decoración main

```
27 void decoprint(){
28     HANDLE hd = GetStdHandle(STD_OUTPUT_HANDLE);
29     SetConsoleTextAttribute(hd, 14);
30
31     for(int a=0;a<120;a++){
32
33         gotoxy(a,29);
34         printf("%c",177);
35
36         if(a<30){
37             gotoxy(0,a);
38             printf("%c",177);
39
40             fflush(stdin);
41
42             gotoxy(119,a);
43             printf("%c",177);
44         }
45
46         gotoxy(a,0);
47         printf("%c",177);
48     }
49
50     gotoxy(50,13);
51     SetConsoleTextAttribute(hd, 11);
52 }
```

Ilustración 54 Decoración

```
54 void PrincipioMenu(){
55     HANDLE hd = GetStdHandle(STD_OUTPUT_HANDLE);
56     SetConsoleTextAttribute(hd, 11);
57     for(int a=0;a<120;a++){ //Bordes
58
59         gotoxy(a,29);
60         printf("%c",207);
61
62         if(a<30){
63             gotoxy(0,a);
64             printf("%c",207);
65
66             fflush(stdin);
67
68             gotoxy(119,a);
69             printf("%c",207);
70         }
71
72         gotoxy(a,0);
73         printf("%c",207);
74     }
75     SetConsoleTextAttribute(hd, 21);
76     gotoxy(50,3);
77     cout<<"Bienvenido cliente"<<endl;
78     SetConsoleTextAttribute(hd, 13);
79     for(int c=5;c<10;c++){ //Letras en y
80         gotoxy(35,c);
81         printf("%c",186);
82
83         gotoxy(45,c);
84         printf("%c",186);
85
86         gotoxy(50,c);
87         printf("%c",186);
88
89         gotoxy(53,c);
90         printf("%c",186);
91
92         gotoxy(59,c);
93         printf("%c",186);
94
95         gotoxy(62,c);
96         printf("%c",186);
```

```
96     printf("%c",186);
97
98     gotoxy(69,c);
99     printf("%c",186);
100
101     gotoxy(75,c);
102     printf("%c",186);
103
104
105     for(int d=36;d<42;d++){ //B
106         gotoxy(d,5);
107         printf("%c",205);
108
109         gotoxy(d,9);
110         printf("%c",205);
111
112         gotoxy(d,7);
113         printf("%c",205);
114
115     }
116
117     for(int AA=46;AA<50;AA++){ //A
118         gotoxy(AA,5);
119         printf("%c",205);
120
121         gotoxy(AA,7);
122         printf("%c",205);
123
124         // individuales
125         gotoxy(42,6); //B
126         printf("%c",186);
127         gotoxy(42,8);
128         printf("%c",186);
129
130     int p=54,q=5; //N
131     while(p!=59){
132         gotoxy(p,q);
133         printf("%c",187);
134         p++;
135         q++;
136     }
137
138     for(int e=63;e<66;e++){ //C
```

```
135     } q++;
136
137     for(int e=63;e<66;e++){ //C
138         gotoxy(e,5);
139         printf("%c",205);
140         gotoxy(e,9);
141         printf("%c",205);
142     }
143
144     for(int o=70;o<75;o++){ //O
145         gotoxy(o,5);
146         printf("%c",205);
147         gotoxy(o,9);
148         printf("%c",205);
149     }
150
151     // Esquinas Pulidas
152     gotoxy(35,5); //B
153     printf("%c",201);
154     gotoxy(35,9);
155     printf("%c",200);
156     gotoxy(35,7);
157     printf("%c",204);
158
159     gotoxy(45,5); //A
160     printf("%c",201);
161     gotoxy(50,5);
162     printf("%c",187);
163     gotoxy(45,7);
164     printf("%c",204);
165     gotoxy(50,7);
166     printf("%c",185);
167
168     gotoxy(53,5); //N
169     printf("%c",201);
170     gotoxy(59,9);
171     printf("%c",188);
172
173     gotoxy(58,9);
174     printf("%c",200);
175
176     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

ListaDeListasCli
ListaDeusuarios

Ilustración 57 Archivos de lista de listas

14642
15252
20812
31041
32118
32932
33097
37643
39318
41224

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).

ListaDeListasCli: Bloc de notas

Archivo Edición Formato Ver Ayuda

41224

31041

32118

20812

14642

Ilustración 59 Archivo lista de listas de cliente

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	5227	Prueba	Lista	2147483647	Manuel	2147483647	8.9469E+15	Ahorro	Activo	20/11/2022	Deudor	7000	0	10	700	
2	6315	Ricardo	Salazar	4442236658	Jesus	4441187824	5.7244E+15	Cre_hipo	Activo	22/11/2022	Deudor	0	150000	20	30000	Propiedad

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.

ListaDeUsuarios: Bloc de notas

Archivo Edición Formato Ver Ayuda

32932

33097

37643

39318

15252

Ilustración 61 Archivo lista de listas usuario

	A	B	C	D	E	F	G	H	I	J	K
1	6252	Fernando	Hernandez	24	4448897851	fer@hotmail	AvJuarez	Fersito	152164	Activo	Ejecutivo

Ilustración 60 Archivo lista usuarios1