UNIVERSIDAD LA SALLE

FACULTAD DE INGENIERÍA

PROGRAMACIÓN ESTRUCTURADA

PROYECTO FINAL “BANCO VAAD”

PROFESOR: Alejandro Lara Villareal

GRUPO: 205

INTEGRANTES:

Moran Franco Alethia Silvana

Cadena Aguilar Vania

González Robles David Alejandro

Lopez Vudoyra Angel

MANUAL

Bibliotecas gráficas ocupadas:

-Graphics.h

Pasos para instalar Graphics.h:

1. Descargue CodeBlocks (codeblocks-20.03-setup.exe) sin el compilador GNU GCC de MinGW, un compilador TDM-GCC separado y los binarios WinBGIm de la sección de descargas a continuación .

2. Si ya había instalado CodeBlocks (codeblocks-20.03mingw-setup.exe) que viene con el compilador, le recomiendo desinstalarlo.

3. Instale el compilador TDM-GCC durante la instalación, debe seleccionar MinGW / TDM (32 bits) y asegurarse de que la opción Agregar a la ruta esté seleccionada. Hace que CodeBlocks detecte automáticamente el compilador.
   LINK DEL COMPILADOR: TDM-GCC: https://jmeubank.github.io/tdm-gcc/download/

4. Instale CodeBlocks (codeblocks-20.03-setup.exe) . Cuando abra CodeBlock, detectará automáticamente el compilador. CodeBlocks almacena en caché la configuración del compilador anterior, por lo que para asegurarse de que la configuración del compilador funcione, vaya a Configuración> Compilador y luego restablezca los valores predeterminados . Ahora detectará TDM-GCC como compilador predeterminado si aún no lo ha detectado.

5. Ahora es el momento de colocar los binarios descargados en la ubicación correcta. Copie los archivos dentro de lib (libbgi.a) e incluya la carpeta (graphics.h& winbgim.h) en la carpeta respectiva del compilador TDM-GCC C:\TDM-GCC-32\lib& C:\TDM-GCC-32\include.
   Puede acceder a los documentos por medio del siguiente link: https://drive.google.com/open?id=1joDbQBIsnjCElEpUx4z59oWbiXcwACha

6. Ahora en CodeBlocks dentro de Ajustes> Ajustes del compilador> Linker , añaden libbgi.aen Link Libraries: y pusieron -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32en Otras opciones del vinculador:

7. Guarde la configuración y estará listo para ejecutar su programa de gráficos en CodeBlocks.

Estructuras utilizadas en el proyecto

Enseguida se presentarán las estructuras utilizadas para la realización del proyecto junto a una breve explicación de su función así como una imagen de su visualización en el programa ejecutado.

➔ En el archivo en c.

struct persona

Estructura persona o también nombrada P, funge como estructura principal en donde se almacenan todos los datos personales de cada usuario, así como también datos específicos necesarios para el funcionamiento del sistema empresarial, además de contar con las demás estructuras que serán ocupadas a lo largo del programa.

```
////////////////////////////////////
struct persona
{
  //Datos generales
    N nombre;
    int edad;
    int edo_civil;
    D direccion;
    F nacimiento;
    char rfcc[11];
    //Datos especificos
    T tarjeta;
    char pass[10];
    O operacion[20];
};typedef struct persona P;
////////////////////////////////////
```

struct nombres

Estructura nombres o también llamada N, dicha estructura está empleada para almacenar el nombre completo de los usuarios.

```
////////////////////////////////////
struct nombres
{
    char np[20];
    char app[20];
    char apm[20];
};typedef struct nombres N;
////////////////////////////////////
```

struct direcc

Estructura direcc o también nombrada D, esta estructura se encarga de almacenar los datos que le proporcione el usuario que en conjunto conforman su dirección de residencia.

```
////////////////////////////////
struct direcc
{
    char calle[20];
    int num;
    char col[20];
};typedef struct direcc D;
////////////////////////////////
```

Struct fecha

Estructura fecha o también llamada F, dicha estructura tiene la función de almacenar la fecha de nacimiento proporcionado por el usuario

```
////////////////////////////////////
struct fecha
{
    char dia[3];
    char mes[3];
    char anio[5];
};typedef struct fecha F;
////////////////////////////////////
```

struct cuentas

La estructura cuentas o también nombrada T, es la encargada de almacenar los datos de cuenta de cada usuario, tal les como su número de cuenta, el valor del saldo y crédito que ésta posea así como también el tipo de tarjeta con la que cuenta cada usuario.

```
////////////////////////////////////
struct cuentas
{
    int numcta[5];
    float saldo, cred;
    char tipo[20];
};typedef struct cuentas T;
////////////////////////////////////
```

struct operaciones

Estructura operaciones o también nombrada O, en caso de que el usuario haya deseado realizar un ingreso o egreso de dinero, dicha función se encarga de almacenar qué tipo de movimiento fue, con qué fin y cuál fue el monto ocupado.

```
struct operaciones
{
    float monto;
    int egoin, tipo;
};typedef struct operaciones O;
////////////////////////////////
```

➔ En el archivo cpp.

```
struct oper
```

Para poder ejecutar la librería Graphics.h, se tuvo que crear otra estructura en donde se almacenaron los valores obtenidos del programa en c en el archivo nombrado como cuentas.csv en excel, esta estructura dispone de las mismas variables y datos que la estructura operaciones del primer programa; nombrada estructura oper o también R.

```
struct oper
{
    float monto;
    int egoin, tipo;
};typedef struct oper R;
```

Variables principales del programa

En este apartado se encuentran las variables más importantes ocupadas a lo largo del programa, al igual que una concisa explicación sobre su utilidad y en que parte del programa puedes encontrarlas.

En el archivo en c. :

- **char pass:** Esta variable contiene la contraseña decidida por el usuario, su importancia recae en que, es una de las especificaciones requeridas para que cada usuario pueda entrar a su cuenta. Se encuentra en la estructura persona.
- **int numcta :** Esta variable dicta el número de cuenta de cada usuario, es indispensable ya que funge como uno de los requisitos para que el usuario pueda ingresar a su cuenta, esta variable es llenada con números aleatorios sin posibilidad de modificación por parte del usuario. Se ubica en la estructura cuentas.
- **float saldo:** Dicha variable contiene el saldo del usuario, este rubro es llenado por el mismo sin embargo el valor puede modificarse cada vez que el usuario haga algún movimiento en su cuenta tal como ingresar o egresar dinero. Su ubicación se encuentra en la estructura cuentas.
- **float cred:** Esta variable empieza a actuar si el usuario decide retirar más dinero que el que tiene en su saldo, ya que activa la opcion de credito en donde se le descontará

el monto que decida de su saldo actual, dando así valores a esta variable, que simboliza la deuda que lleva a con el banco. Esta variable se encuentra en la estructura cuentas.

- **int egoin:** En esta variable se encuentra la decisión que toma el usuario con respecto al movimiento de su cuenta, si es que quiere transferir, ingresar, egresar, etc… algún valor de su saldo actual. Se ubica en la estructura operaciones.
- **float monto:** Esta variable posee el valor con el que el saldo del usuario puede verse afectado respecto a la decisión que haya llenado en *int egoin,* el valor que el usuario decida agregar a esta variable será el total que se le agregara o descontará de su saldo actual. Esta variable tiene ubicación en la estructura operaciones.
- **int usuario:** Dicha variable contará con el valor del usuario que se encuentre ejecutando el programa en el momento, de esa forma se valida su existencia en el programa y solo se abrirá su cuenta registrada con anterioridad. La ubicación de esta variable se encuentra en el programa principal int main(void).

En el archivo cpp.

Para poder crear las gráficas correspondientes al estado de cuenta del usuario se usaron varias variables de contador.

- **int valida, ti1, ti2, ti3, ti4, te1, te2, te3:** La variable valida cuenta como el 100 por ciento del total del estado de cuenta, mientras que todos los ti fungen como contadores de los movimientos, estos por una regla de tres se vuelven coordenadas que al momento de ponerlas en las funciones de Graphics.h, crean las barras correspondientes

Archivos

Para el proyecto se hizo uso de un archivo cuentas.csv en excel para presentar al cliente sus ingresos y egresos y tenga un mejor manejo de su acumulado. Así mismo se ocupa este mismo archivo en otro programa vinculado al del banco en .ccp con la biblioteca de graphics.h para que el cliente vea sus ingresos y egresos representados en gráfica de barras.

Funciones utilizadas para el proyecto

A continuación se presentan las funciones que usamos para el proyecto y una imagen adjunta de cómo se ve en el programa:

```
int main (programa de datos)
```

Función principal del programa, de aquí parten todas las funciones explicadas posteriormente, aquí se le presentan al usuario todos los menús principales, crear cuenta, ingresar o salir del banco y llena los datos predeterminados. Si va a crear cuenta manda a llamar a la función `llena.`

Cuando el usuario ya ingresa con su debida clave y número de cuenta manda a la función `validaingreso.` Después despliega el nombre, saldo total, crédito a pagar y el menú secundario "Mostrar mis datos" que va a la función `imprimedatos`, "Registrar mi ingreso o egreso" que manda al usuario a la función `llenaop,` "Pagar crédito" que valida si se cuenta con el monto adecuado para pagar los adeudos del crédito, en caso negativo despliega un mensaje que aún no se cuenta con los fondos necesarios para realizar el pago y la opción de pagar con todo su saldo actual.

También se encuentra la opción "Ver mi estado de cuenta" que va a `edocuenta,` funcion explicada más adelante; por último tenemos la opción de hacer un llenado predeterminado para la presentación del proyecto con su respectiva función `autollenado` finalmente está la opción de salir de la cuenta sin embargo aún puede ingresar un usuario diferente.

```
void  main (programa de gráficos)
```

Este programa lee los archivos guardados del estado de cuenta y los guarda en la estructura del programa, después realiza una regla de tres con todos los datos de un mismo tipo de acción para calcular el porcentaje, de acuerdo al porcentaje obtenido se asignan coordenadas para graficar por medio de barras las acciones con respecto al total de ingresos y egresos con su respectivo tipo.

```
void inicia
```

La función se usa al empezar el programa e imprime las palabras "*Banco Vaad*" por medio de la impresión de caracteres de una matriz diseñada con ocho renglones y 21 columnas; no recibe ni devuelve nada.

```
void inicia(void){
    char w=223;
    char b[8][21]={' '};
    int i,j;

b[0][0]=w,b[0][1]=w,b[0][5]=w,b[0][8]=w,b[0][11]=w,b[0][14]
=w,b[0][15]=w,b[0][18]=w,b[0][19]=w;
b[1][0]=w,b[1][2]=w,b[1][4]=w,b[1][6]=w,b[1][8]=w,b[1][9]=w,
b[1][11]=w,b[1][13]=w,b[1][17]=w,b[1][20]=w;
b[2][0]=w,b[2][2]=w,b[2][4]=w,b[2][6]=w,b[2][8]=w,b[2][9]=w,
b[2][11]=w,b[2][13]=w,b[2][17]=w,b[2][20]=w;
b[3][0]=w,b[3][1]=w,b[3][4]=w,b[1][5]=w,b[3][6]=w,b[3][8]=w,
b[3][9]=w,b[3][11]=w,b[3][13]=w,b[3][17]=w,b[3][20]=w;
b[4][0]=w,b[4][2]=w,b[4][4]=w,b[4][5]=w,b[4][6]=w,b[4][8]=w,
b[4][10]=w,b[4][11]=w,b[4][13]=w,b[4][17]=w,b[4][20]=w;
b[5][0]=w,b[5][2]=w,b[5][4]=w,b[5][6]=w,b[5][8]=w,b[5][10]
=w,b[5][11]=w,b[5][13]=w,b[5][17]=w,b[5][20]=w;
b[6][0]=w,b[6][2]=w,b[6][4]=w,b[6][6]=w,b[6][8]=w,b[6][11]
=w,b[6][13]=w,b[6][17]=w,b[6][20]=w;
b[7][0]=w,b[7][1]=w,b[7][4]=w,b[7][6]=w,b[7][8]=w,b[7][11]
=w,b[7][14]=w,b[7][15]=w,b[7][18]=w,b[7][19]=w;

for(i=0;i<8;i++)
{
    for(j=0;j<21;j++)
    {
        printf("%c ",b[i][j]);
    }
    printf("\n");
}
```

```
int validaingreso
```

Recibe un arreglo de estructura tipo personas y un entero, esta función sirve para preguntar los datos de ingreso de cuenta, usuario y busca en toda la estructura el usuario ingresado, si no existe dicho usuario se activará una bandera que desplegará el mensaje de "Usuario incorrecto" ; en caso de que el usuario exista solicitará una contraseña que también activará una bandera si es incorrecta y pedirá nuevamente al cliente su número de cuenta y contraseña. Si el ingreso es correcto regresa un entero el cual se guardará en la variable "usuario" como posición actual del cliente dentro de la estructura *Personas.*

```
do
{
    v1=0;
    v2=0;
    band=0;
    printf("Dame tu usuario\n");
    scanf("%d", &num1);
    for(dir2=0;dir2<dir && band==0;dir2++){
        aux=0;
        aux=((personas[dir2].tarjeta.numcta[0])*(10000
        +((personas[dir2].tarjeta.numcta[1])*(1000))+(
        (personas[dir2].tarjeta.numcta[2])*(100))+(
        (personas[dir2].tarjeta.numcta[3])*(10))+(
        (personas[dir2].tarjeta.numcta[4])*(1));
        if(num1!=aux)
        {
            v1++;
        }
        else{
            printf("Usuario correcto\tDame tu
            contrase%ca\n", 164);
            scanf("%s", &pass);
            if(strcmp(personas[dir2].pass, pass)==0){
                v2++;
                band=1;
            }
            else
                printf("La contraseña es incorrecta\n"
            }
    }
```

void llena

No regresa nada ya que trabaja por medio de referencia, recibe una estructura de tipo personas y un apuntador de dirección, dependiendo de la dirección pide datos del usuario y , nombre, apellidos, edad, fecha de nacimiento, estado civil, dirección y datos de la cuenta bancaria así como una contraseña tipo *char,* y los llena en el vector de estructuras en la posición *dir a menos que el vector esté lleno y no pueda recibir más usuarios.

```
void llena (P personas[10],int *dir)
{
    int i, w=223;
    char pass[20];
    if (*dir <10)
    {
        printf("---INGRESA LOS DATOS REQUERIDOS---\n");
        printf("\n\tNombre Propio\n");
        fflush(stdin);
        gets(personas[*dir].nombre.np);
        printf("\tApellido Paterno\n");
        fflush(stdin);
        gets(personas[*dir].nombre.app);
        printf("\tApellido Materno\n");
        fflush(stdin);
        gets(personas[*dir].nombre.apm);
        printf("\tEdad\n");
        scanf("%d",&personas[*dir].edad);
        printf("\tEdo civil\n");
        fflush(stdin);
        gets(personas[*dir].edo_civil);
        printf("\n\tDIRECCION\nCalle\n");
        fflush(stdin);
        gets(personas[*dir].direccion.calle);
        printf("\tNumero\n ");
        scanf("%d",&personas[*dir].direccion.num);
        printf("\tColonia\n ");
        fflush(stdin);
        gets(personas[*dir].direccion.col);
```

Para confirmar contraseña ingresada correctamente, debe tener los siguientes requerimientos

-8 caracteres

-Al menos una mayúscula

-Al menos una minúscula

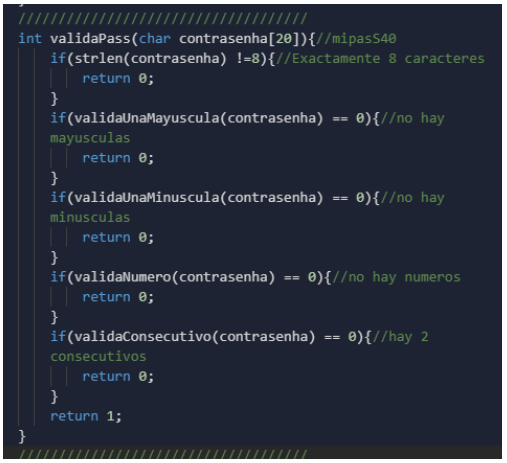
-Mínimo un número

-No más de dos caracteres consecutivos

int validaPass

Función más importante para validar la contraseña de un nuevo usuario, recibe dicha contraseña y la envía a diferentes funciones dentro de ella misma las cuales validaron por

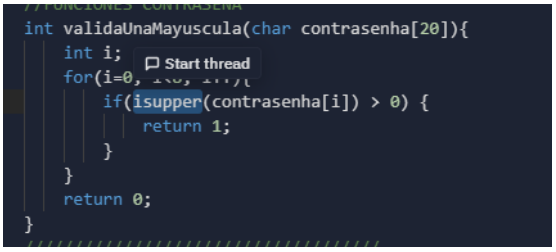
separado que cumpla con las condiciones indicadas con anterioridad; una vez evaluadas regresaran un entero entre cero y uno; posteriormente en esta función por medio de condicionales se regresará uno a la función principal si cumple con al menos una mayúscula, al menos una minúscula, un número y no más de dos caracteres consecutivos, en caso de que no se cumpla al menos con una de los requerimientos regresará cero y la función llena no dejará de preguntar por una contraseña segura hasta que *validaPass* regrese uno.



```
////////////////////////////////
int validaPass(char contrasenha[20]){//mipasS40
    if(strlen(contrasenha) !=8){//Exactamente 8 caracteres
        return 0;
    }
    if(validaUnaMayuscula(contrasenha) == 0){//no hay
    mayusculas
        return 0;
    }
    if(validaUnaMinuscula(contrasenha) == 0){//no hay
    minusculas
        return 0;
    }
    if(validaNumero(contrasenha) == 0){//no hay numeros
        return 0;
    }
    if(validaConsecutivo(contrasenha) == 0){//hay 2
    consecutivos
        return 0;
    }
    return 1;
}
////////////////////////////////
```

int validaUnaMayuscula

La contraseña ingresada por el usuario pasa a validar caracter por caracter si se encuentra una mayúscula con ayuda de la función *"isupper(contrasenha)>0"* en caso de ser positivo regresa un uno.



```
int validaUnaMayuscula(char contrasenha[20]){
    int i;
    for(i=0,
        if(isupper(contrasenha[i]) > 0) {
            return 1;
        }
    }
    return 0;
}
```

int validaUnaMinuscula

Esta función posee un funcionamiento similar a *validaUnaMayuscula* con ayuda de un ciclo *for* se revisa cada casilla de la cadena de texto *pass[20]; islower(contrasenha[i]) > 0)* regresará uno en caso de que se encuentre una letra minúscula.

```
////////////////////////////////////
int validaUnaMinuscula(char contrasenha[20]){
    int i;
    for(i=0; i<8; i++){//0 1 2 3 4 5 6 7
        if(islower(contrasenha[i]) > 0) {
            return 1;
        }
    }
    return 0;
}
////////////////////////////////////
```

nt validaNumero

Recibe una cadena de máximo 20 letras reconocida como pass; revisa caracter a caracter que exista un número del cero al nueve, en caso de tener un número devuelve uno a la función *ValidaPass;* en caso contrario devuelve cero (quiere decir que no se encuentra un número en la contraseña).

```
////////////////////////////////////
int validaNumero(char pass[20]){
    int i;
    for(i=0; i<8; i++){//0 1 2 3 4 5 6 7
        if(pass[i] >= '0' && pass[i]<='9'){
            return 1;
        }
    }
    return 0;
}
////////////////////////////////////
```

.

int validaConsecutivo

Función que recibe la contraseña ingresada por el usuario para crear una cuenta; revisa elemento a elemento de la cadena que la resta de la casilla siguiente y la que se está evaluando sea menos uno, es decir, existen caracteres consecutivos ya sea numérico o alfabético y por lo tanto devuelve un cero a la función *validaPass*.

```
////////////////////////////////////
int validaConsecutivo(char pass[20]){
    // mmP12345
    int i;
    for(i=0; i<8-1; i++){//0 1 2 3 4 5 6 7
        if(pass[i] - pass[i+1] == -1){// A B --> 65 66 ==-1
        //if(pass[i]+1 == pass[i+1]){
        //if(pass[i+1] - pass[i] == 1){
            return 0;
        }
    }
    return 1;
}
////////////////////////////////////
```

int imprimeResultado

Ubicado después de *validaPass* en la función llenar recibe la misma contraseña ingresada por el usuario y vuelve a verificar la contraseña y dependiendo de lo que regrese la función *validaPass* si regresa uno es porque cumple con todos los requisitos e imprime "Tu password es seguro y válido" si no imprime "Tu password es inseguro y no es válido" para que el usuario sepa por qué le pide nuevamente la contraseña, o si la contraseña es segura.

```
int imprimeResultado(char pass[20]){

    if(validaPass(pass) == 1){
        printf("\nTu password es seguro y valido");
    }else{
        printf("\nTu password es inseguro y no es valido");
    }
    return validaPass(pass);
}
////////////////////////////////
```

void sacarfc

Ubicada dentro de la función llena; obtiene el RFC del usuario, recibe dos estructuras una tipo *Fecha* y *Nombre* y una cadena de texto de máximo veinte caracteres, *rfcc*; copia la primera letra del nombre y del apellido, los ultimos dos digitos de la fecha de nacimiento y concatena el mes y el dia en *rfcc.* Esta función no regresa nada.

```
}
void sacarfc(N nombre,F naci,char rfcc[11])
{
    char nom, mat;
    nom=nombre.np[0];
    mat=nombre.apm[0];
   strncpy(rfcc,nombre.app,2);
   //strncat(rfcc,nombre.apm,1);
   //strncat(rfcc,nombre.np,1);
   rfcc[2] =mat;
   rfcc[3] =nom;
   rfcc[4] =naci.anio[2];
   rfcc[5] =naci.anio[3];
   rfcc[6] ='\0';
   strcat(rfcc,naci.mes);
   strcat(rfcc,naci.dia);
   strupr(rfcc);
}
```

void datos

Recibe una estructura de tipo personas y un arreglo tipo char con el rfc generado con anterioridad; no regresa nada, ya que solo imprime los datos guardados del usuario; no imprime los datos bancarios únicamente datos personales del usuario.

```
void datos(P personas, char rfc)
      printf("Hola %s %s %s\n",personas[*dir].nombre.np,
      personas[*dir].nombre.app,personas[*dir].nombre.apm);
        printf("Edad: %s",&personas[*dir].edad);
        printf("Estado civil: %s\n",personas[*dir]
        .edo_civil);
        printf("Direccion: %s no.%d Col. %s\n",personas
        [*dir].direccion.calle,&personas[*dir]
        .direccion.num,personas[*dir].direccion.col);
        printf("Fecha de nacimiento: %s de %s del %s\n",
        personas[*dir].nacimiento.dia,personas[*dir]
        .nacimiento.mes,personas[*dir].nacimiento.anio);
        printf("CURP: %s\n", personas[*dir].rfcc);
```

void fijas

Para realizar transferencias hemos agregado datos predeterminados para algunas personas con ayuda de la función fijas, recibe el arreglo de personas y el apuntador de la dirección tipo entero; no regresa nada pues las operaciones las realiza por referencia.

Datos de los usuarios ya registrados:

Persona 1: Andres Hernandez Sada
Contraseña: L7jhht6m
Num tarjeta: 96024
Persona 2: Ricardo Guitierrez Espinal
Contraseña: P8k02Pn4
Num tarjeta: 05793
Persona 3: Paulo Salcedo Ramirez
Contraseña: V8m2hlp1
Num tarjeta: 93162
Persona 4: Carlos Servin Blanco
Contraseña: Ouy72mp0
Num tarjeta: 73541
Persona 5: Roberto Romero Marin
Contraseña: 7jY8mlo2
Num tarjeta: 37240

void llenaop

Esta función es ocupada para registrar las acciones del usuario egresos e ingresos y recibe la estructura *Personas,* la posición del usuario actual donde se han guardado todos los datos, un apuntador de dirección tipo entero y otro apuntador tipo entero para contabilizar el número de acciones en total *(*m)*, el programa puede realizar máximo veinte, y no devuelve nada al *main*; por medio de un *switch* se escanea la acción que el usuario desea realizar.

En caso de ser ingreso pide una opción como concepto por medio de un *switch* con las opciones de Laboral(bono, pagos de la empresa, etc), ventas, servicios, u otros y una vez seleccionado el concepto el programa pide monto para agregarlo al acumulado de la cuenta del usuario.

```
}
void llenaop(P personas[10], int usuario, int *dir, int *m){
  int aux, aux2, i,band, valida;
  printf("Selecciona\n 1)ingreso  2)Egreso\n");
  scanf("%d", &personas[usuario].operacion[*m].egoin);
  switch(personas[usuario].operacion[*m].egoin)

  {
    case 1: printf("Selecciona el tipo de ingreso\n1)Laboral
    (sueldo,honorario,bono)\n2)Ventas\n3)Servicios\n4)
    Otros\n");
            scanf("%d", &personas[usuario].operacion[*m]
            .tipo);
            printf("Monto a ingresar\n");
            scanf("%f", &personas[usuario].operacion[*m]
            .monto);
            //printf("%f", personas[usuario].operacion[*m]
            .monto);
            personas[usuario].tarjeta.saldo=(personas
            [usuario].tarjeta.saldo)+(personas[usuario]
            .operacion[*m].monto);
            (*m)++;
            break;
```

Para los egresos el usuario tiene diferentes opciones para este: gastos, retiros o transferencias los cuales tienen operaciones diferentes dependiendo del *switch* correspondiente..

```
{
 case 1:
   printf("Cuanto desea retirar?\n");
   scanf("%f", &personas[usuario].operacion[*m]
   .monto);
   //printf("%f", personas[usuario].operacion
   [*m].monto);
   if(personas[usuario].tarjeta.saldo<personas
   [usuario].operacion[*m].monto)
   {
     printf("Tu cuenta no cuenta con saldo
     suficiente\n¿Deseas agregar el monto
     faltante de tu cuenta de credito? 1)Si otro
     No\n");
     scanf("%d", &aux);
     if(aux==1)
     {
       personas[usuario].tarjeta.cred=(personas
       [usuario].operacion[*m].monto)-(personas
       [usuario].tarjeta.saldo);
       personas[usuario].tarjeta.saldo=0;
       (*m)++;
     }
   }
```

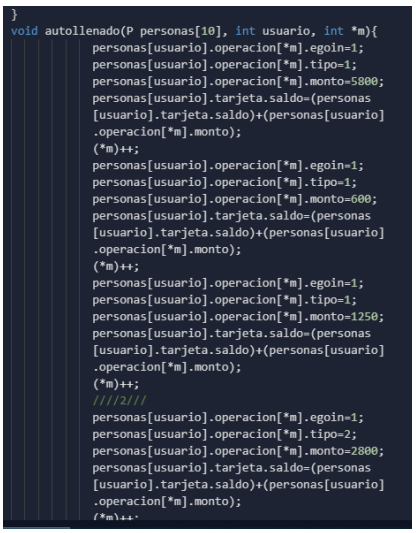
En las transferencias se pide un monto y un número de tarjeta a donde se va a agregar el monto y se descuenta del acumulado del usuario, por medio de un ciclo *for* busca si el número de cuenta existe, en caso afirmativo cambia una bandera y guarda la posición donde se encuentra el número de cuenta en una variable auxiliar para usar esa misma dirección y agregar el monto ingresado con anterioridad.

Para los retiros simples y gastos por definir si el monto a salir es mayor a lo que posee el usuario se abre un mensaje indicando insuficiencia de fondos y despliega otro menú para preguntar si se desea agregar el monto faltante a una cuenta de crédito la cual tendrá que pagar de sus ingresos; si tiene suficiente saldo solo se le resta al acumulado del cliente.

```
 case 3: printf("Dame el monto\n");
   scanf("%f", &personas[usuario].operacion[*m]
   .monto);
   //printf("%f", personas[usuario].operacion
   [*m].monto);
   if(personas[usuario].tarjeta.saldo<personas
   [u  Start thread  cion[*m].monto)
   {
     printf("Tu cuenta no cuenta con saldo
     suficiente\n¿Deseas agregar el monto
     faltante de tu cuenta de credito? 1)Si otro
     No\n");
     scanf("%d", &aux);
     if(aux==1)
     {
       personas[usuario].tarjeta.cred=(personas
       [usuario].operacion[*m].monto)-(personas
       [usuario].tarjeta.saldo);
       personas[usuario].tarjeta.saldo=0;
       (*m)++;
     }
   }
   else{
     personas[usuario].tarjeta.saldo=(personas
     [usuario].tarjeta.saldo)-(personas[usuario]
     .operacion[*m].monto);
     (*m)++;
   }
     break;
```

```
void autollenado
```

Con la finalidad de economizar tiempo en la presentación de proyecto se añadió una función para autollenar los ingresos y egresos de manera predeterminada; esta función recibe un arreglo de estructura de tipo personas con diez espacios, la posición del usuario "int usuario", y dos punteros de tipo entero uno con el contador de las operaciones de ingresos y egresos "m"; llena tipo de operación retiro, laboral, etc de acuerdo al *switch* de la función de operaciones una vez introducidos los datos aumenta el contador de operaciones.



```
}
void autollenado(P personas[10], int usuario, int *m){
            personas[usuario].operacion[*m].egoin=1;
            personas[usuario].operacion[*m].tipo=1;
            personas[usuario].operacion[*m].monto=5800;
            personas[usuario].tarjeta.saldo=(personas
            [usuario].tarjeta.saldo)+(personas[usuario]
            .operacion[*m].monto);
            (*m)++;
            personas[usuario].operacion[*m].egoin=1;
            personas[usuario].operacion[*m].tipo=1;
            personas[usuario].operacion[*m].monto=600;
            personas[usuario].tarjeta.saldo=(personas
            [usuario].tarjeta.saldo)+(personas[usuario]
            .operacion[*m].monto);
            (*m)++;
            personas[usuario].operacion[*m].egoin=1;
            personas[usuario].operacion[*m].tipo=1;
            personas[usuario].operacion[*m].monto=1250;
            personas[usuario].tarjeta.saldo=(personas
            [usuario].tarjeta.saldo)+(personas[usuario]
            .operacion[*m].monto);
            (*m)++;
            ////2///
            personas[usuario].operacion[*m].egoin=1;
            personas[usuario].operacion[*m].tipo=2;
            personas[usuario].operacion[*m].monto=2800;
            personas[usuario].tarjeta.saldo=(personas
            [usuario].tarjeta.saldo)+(personas[usuario]
            .operacion[*m].monto);
```

```
void edocuenta
```

La función recibe la estructura de diez personas, la variable entera de la posición del usuario que ingresó, el contador de las operaciones y un apuntador para abrir un archivo.

Primero la función abre el archivo para sobreescribirlo, después imprime con un ciclo una a una las acciones realizadas y registradas en *"llenaop"* según el tipo y el monto, después imprime de la misma manera los egresos, el tipo y el monto retirado. Para finalizar imprime el saldo disponible, el crédito a pagar en caso de tener y mediante una resta de los valores si dicha resta es menor a cero se le mostrará al cliente un mensaje informando que tiene saldo a pagar o en caso contrario si su saldo está a favor, es decir, no tiene deudas con el banco y por último cierra el archivo con los datos de los ingresos y egresos.

```
--------------------\n");
for(i=0;i<(*m);i++){
    if(personas[usuario].operacion[i].egoin==1){
        switch(personas[usuario].operacion[i].tipo)
        {
            case 1: printf("Laboral\t");
                    break;
            case 2: printf("Ventas\t");
              break;
            case 3: printf("Servicios\t");
            break;
            case 4: printf("Otros\t");
            break;
        }
        printf("%f\n", personas[usuario].operacion
        [i].monto);
        fprintf(archivo, "%d %d %f\n", personas
        [usuario].operacion[i].egoin, personas
        [usuario].operacion[i].tipo, personas
        [usuario].operacion[i].monto);
    }
}
```