Struct

```
struct Datos {
    int nroCuenta;
    int limiteExtraccion;
    int saldoActual;
    int saldoInicial;
    int clave;
    int contExtracciones;
};
```

Define una estructura llamada Datos que almacena información sobre una cuenta bancaria, incluyendo el número de cuenta, el límite de extracción diario, el saldo actual, el saldo inicial, la clave de acceso y un contador de extracciones.

Prototipo de funciones

```
// Prototipos de funciones
void claveDigitos(int&);
void cargarDatos(Datos[], int, int&);
void imprimir(Datos[], int);
double promedio(Datos[], int);
int validarNroCuenta(Datos[], int, int);
bool validarDigitos(int);
bool validarClave(int, int);
void realizarExtraccion(Datos[], int, int);
void realizarDeposito(Datos[], int, int);
bool extraccionesLimite(Datos[], int);
void cantidadCuentas(Datos[], int, int&, int&, int&);
```

Estos son los prototipos de las funciones que se utilizarán más adelante en el programa.

Funcion main

Datos cuentas[DIM];

- está creando un arreglo de estructuras llamado `cuentas`, donde cada elemento del arreglo es una instancia de la estructura `Datos`.

En resumen, al declarar Datos cuentas[DIM];, estás reservando un espacio en memoria para almacenar datos de múltiples cuentas bancarias, lo que te permite trabajar con ellas de manera eficiente y estructurada. Cada elemento del arreglo representa una cuenta individual y contiene los datos de esa cuenta, como el número de cuenta, el saldo, la clave, etc.

```
    cout<<"Hay cuentas para cargar?(Si=1/No=0) "<<endl;
    cin>>resp;

    while((resp!=0)&&(resp!=1)){

        cout<<"Respuesta mal ingresada(Si=1/No=0), por favor vuelva a
ingresarla"<<endl;
        cin>>resp;
    }
```

Parte del programa que se encarga de iniciarlo, verificando si hay o no datos a cargar

El ciclo while, se encarga de validar la respuesta para que se `SI:1` o `NO:0`

Llamado de la funcion dentro del main

Funcion cargar datos

```
cargaDatos(cuentas,resp,dim);
```

Funcion para imprimir los valores ingresados

```
imprimir(cuentas,dim);
```

Movimientos diarios de la cuenta

```
cout<<"INGRESO DE MOVIMIENTOS DIARIOS"<<endl;
    cout<<"Ingrese numero de cuenta: ";
    cin>>resp;
    indiceCuenta=validarNRoCuenta(cuentas,dim,resp);
    while(indiceCuenta==-1){

        cout<<"No se encontro el numero de cuenta por favor vuelva a
ingresarlo: ";
        cin>>resp;

        indiceCuenta=validarNRoCuenta(cuentas,dim,resp);
    }

    while(resp!=0){

        cout<<"Ingrese la clave. ";
        cin>>auxClave;

        claveDigitos(auxClave);
```

```
        n=1;

        while((n>3)||
(validarClave(cuentas[indiceCuenta].clave,auxClave)==false)){
                cout<<"Clave Erronea, vuelva a ingresarla,(intento  "
<<n<<" de 3): "<<endl;
                cin>>auxClave;

                claveDigitos(auxClave);

                n++;
        }


        if(n<=3){

            cout<<"Ingrese la operacion que desea realizar: "<<endl;
            cout<<"E=Extraccion"<<endl;
            cout<<"D=Deposito"<<endl;
            cout<<"C:Consulta de Saldos"<<endl;

            cin>>tpMovimiento;

            while((tpMovimiento!='E')&&(tpMovimiento!='e')&&
(tpMovimiento!='D')&&(tpMovimiento!='d')&&(tpMovimiento!='C')&&
(tpMovimiento!='c')){

                cout<<"ingreso la respuesta erronea, por favor vuelva a
ingresarla: ";
                cin>>tpMovimiento;
            }

            if((tpMovimiento=='E')||(tpMovimiento=='e')){
                cout<<"ingrese el monto que quiere retirar: "<<endl;
                cin>>extraccion;

                if(extraccionesLimite(cuentas,indiceCuenta)==true){
                    extrac(cuentas,indiceCuenta,extraccion);
                }
                else{
                    cout<<"ya supero el limite de extracciones diarias"
<<endl;
                }
            }
            else{

                if((tpMovimiento=='D')||(tpMovimiento=='d')){

                    cout<<"ingrese el monto que quiere depositar: "<<endl;
                    cin>>deposito;

                    depo(cuentas,indiceCuenta,deposito);
```

```
                    cout<<"la operacion se realizo con exito el total de
su cuenta es: $"<<cuentas[i].saldoActual<<endl;
                }
                else{
                    cout<<"Su saldo Actual es de: $"
<<cuentas[i].saldoActual<<endl;
                }
            }

        }
        else{

            cout<<"usted ah usado los 3 intentos, no podra realizarce
ninguna operacion en el dia"<<endl;
        }


    }

    imprimir(cuentas,dim);
```

1ero solicita el nro de cuenta

```
    cout<<"Ingrese numero de cuenta: ";
    cin>>resp;
```

2do valida ese nro de cuenta con la funcion que se encarga de validar los nros de cuenta

```
indiceCuenta=validarNRoCuenta(cuentas,dim,resp);
```

3er si el nro de cuenta es invalido

```
while (indiceCuenta == -1) {
    cout << "No se encontró el número de cuenta, por favor vuelva a
ingresarlo: ";
    cin >> resp;
    indiceCuenta = validarNRoCuenta(cuentas, dim, resp);
}
```

Si indiceCuenta es igual a -1, significa que el número de cuenta no se encontró en el arreglo. Por lo tanto, el código entra en un bucle que permite al usuario ingresar el número de cuenta nuevamente hasta que sea válido:

4to Ingreso a la cuenta si es != de 0

```
while (resp != 0)
```

5to con la funcion claveDigito, valida que sea mayor a 4 digitos y que corresponda

La función claveDigitos se utiliza para asegurarse de que la clave tenga 4 dígitos.

```
        cout<<"Ingrese la clave. ";
        cin>>auxClave;

        claveDigitos(auxClave);
        //Le pasa el parametro que se pidio antes para que sea mandado a
la funcion ejecute los pasos correspondientes

        n=1;
        //El código establece una variable n en 1 para rastrear el número
de intentos de ingreso de clave
```

6to ingreso a un bucle que permite ingresar la clave hasta 3 veces

```
while (n <= 3 && !validarClave(cuentas[indiceCuenta].clave, auxClave)) {
    // ...
}
```

Una vez validada la clave, el usuario podra ingresar para realizar 3 tipos de operaciones

1. E: Extraciones
2. D: Deposito
3. C: Consulta de saldo

`tpMovimiento`, en esta variable es alacenado el valor ingresado

7mo Validacion del tipo de operacion

```
while ((tpMovimiento != 'E' && tpMovimiento != 'e' && tpMovimiento != 'D'
&& tpMovimiento != 'd' && tpMovimiento != 'C' && tpMovimiento != 'c')) {
    cout << "Ingresó una respuesta incorrecta, por favor vuelva a
ingresarla: ";
    cin >> tpMovimiento;
}
```

Se lleva un registro de los intentos de extracción

```
if ((tpMovimiento == 'E' || tpMovimiento == 'e')) {
    cout << "Ingrese el monto que quiere retirar: " << endl;
    cin >> extraccion;

    if (extraccionesLimite(cuentas, indiceCuenta)) {
        realizarExtraccion(cuentas, indiceCuenta, extraccion);
        `cuentas[indiceCuenta].contExtracciones++; `// Registro de intento
de extracción
    } else {
        cout << "Ya superó el límite diario de extracción" << endl;
    }
}
```

Extraccion

```
    if((tpMovimiento=='E')||(tpMovimiento=='e')){
        cout<<"ingrese el monto que quiere retirar: "<<endl;
        cin>>extraccion;

        if(extraccionesLimite(cuentas,indiceCuenta)==true){
            extrac(cuentas,indiceCuenta,extraccion);
        }
        else{
            cout<<"ya supero el limite de extracciones diarias"<<endl;
            }
    }
```

`if((tpMovimiento=='E')||(tpMovimiento=='e'))`

- Esta condición verifica si el usuario eligió realizar una extracción, ya sea escribiendo 'E' o 'e' (mayúsculas o minúsculas)
- Si la condición es verdadera, el programa solicita al usuario ingresar el monto que desea retirar.
- Luego, verifica si el usuario ha superado el límite diario de extracción llamando a la función `extraccionesLimite(cuentas, indiceCuenta)`.
- Si el límite no se ha alcanzado, procede a realizar la extracción llamando a la función `extrac(cuentas, indiceCuenta, extraccion)`
- La cantidad extraída se deduce del saldo de la cuenta y se muestra un mensaje de éxito.
- Si el límite de extracciones diarias se ha superado, se muestra un mensaje de error.

Deposito

```
else{
    if((tpMovimiento=='D')||(tpMovimiento=='d')){
        cout<<"ingrese el monto que quiere depositar: "<<endl;
        cin>>deposito;

        depo(cuentas,indiceCuenta,deposito);
```

```
        cout<<"la operacion se realizo con exito el total de su cuenta es:
$"<<cuentas[i].saldoActual<<endl;
    }
```

`if((tpMovimiento=='D')||(tpMovimiento=='d')`

- Si la condición es verdadera, el programa solicita al usuario ingresar el monto que desea depositar.
- Llama a la función `depo(cuentas, indiceCuenta, deposito)` para aumentar el saldo de la cuenta. Se muestraun mensaje de éxito y se muestra el saldo actual.

Consulta de Saldo

```
    else{
        cout<<"Su saldo Actual es de: $"<<cuentas[i].saldoActual<<endl;
        }
```

- Si el usuario no eligió ni extracción ni depósito, el código simplemente muestra el saldo actual de la cuenta.

Si el usuario ha intentado realizar una operación más de 3 veces (indicado por la variable contExtracciones), se muestra un mensaje que indica que ya ha agotado sus intentos y no podrá realizar más operaciones en el día.

Imprime los nuevos valores con la funcion correspondiente

```
imprimir(cuentas,dim);
```

Llama a la funcion Promedio

```
 cout<<endl<<"El promedio de todas los saldos actuales de las cuentas es:
"<<promedio(cuentas,dim)<<endl;
```

- Imprime un mensaje que indica que se mostrará el promedio de los saldos actuales de todas las cuentas
- Llama a la función promedio(cuentas, dim) para calcular y mostrar el promedio.
- La función promedio toma un arreglo de cuentas y su dimensión como parámetros y devuelve el promedio de los saldos actuales de todas las cuentas.

Tipo de cuentas

Cuentas con saldo acreedor

Cuentas con saldo deudor

Cuentas con saldo nulo

```
    cantidadCuentas(cuentas,dim,contAcreedor,contDeudor,contNulo);

    cout<<"La cantidad de cuentas acreedoras es de: "<<contAcreedor<<endl;
    cout<<"La cantidad de cuentas deudoras es de: "<<contDeudor<<endl;
    cout<<"La cantidad de cuentas nulas es de: "<<contNulo<<endl;
```

`cantidadCuentas(cuentas, dim, contAcreedor, contDeudor, contNulo);`:

- Llama a la función cantidadCuentas para calcular la cantidad de cuentas que son acreedoras, deudoras o nulas.
- Esta función toma el arreglo de cuentas, su dimensión, y tres variables (contAcreedor, contDeudor, y contNulo) por referencia para actualizar sus valores.

Cantidad de extracciones

```
    for(i=0;i<dim;i++){

        if(cuentas[i].contExtraccion==3){
            contExtraccionesCuentas++;
        }

    }

    cout<<"la cantidad de cuentas con 3 extracciones es de: "
<<contExtraccionesCuentas<<endl;
```

- Un bucle for se utiliza para iterar a través de todas las cuentas (índices de 0 a dim - 1) y verificar cuántas de ellas tienen un contExtracciones igual a 3.
- Esto cuenta cuántas cuentas han excedido el límite de extracciones diarias y actualiza la variable contExtraccionesCuentas en consecuencia.

Funciones

Funcion para cargar datos

```
void cargaDatos(Datos cuentas[], int resp, int &dim){

    int i=0;
    int auxNroCuenta, auxClave;

    while(resp==1){

        cout<<"ingrese nro de Cuenta: ";
```

```
        cin>>auxNroCuenta;

        while(auxNroCuenta<=0){
            cout<<endl<<"El numero de cuenta tiene que ser mayor a 0,
vuelva a ingresarlo ";
            cin>>auxNroCuenta;
        }

        cuentas[i].nroCuenta=auxNroCuenta;

        cout<<"Ingrese el monto limite diario para extracciiones: ",
        cin>>cuentas[i].limiteExtraccion;

        cout<<endl<<"Ingrese el saldo incial de la cuenta: ";
        cin>>cuentas[i].saldoIncial;

        cuentas[i].saldoActual=cuentas[i].saldoIncial;

        cout<<endl<<"Ingrese la clave de la cuenta: ";

        cin>>auxClave;

        claveDigitos(auxClave);

        cuentas[i].clave=auxClave;

        cout<<"Hay mas cuentas para cargar?(Si=1/No=0) "<<endl;
        cin>>resp;

        while((resp!=0)&&(resp!=1)){

            cout<<"Respuesta mal ingresada(Si=1/No=0), por favor vuelva a
ingresarla"<<endl;
            cin>>resp;
        }

        cuentas[i].contExtraccion=0;
        i++;
    }

    dim=i;
}
```

Esta función recibe tres argumentos:

- `cuentas[]`: Un arreglo de estructuras de tipo Datos, que se utiliza para almacenar información sobre las cuentas.
- `resp`: Un entero que representa la respuesta del usuario sobre si desea cargar más cuentas.
- `dim`: Una referencia a un entero que se utilizará para almacenar el número total de cuentas cargadas.

La funcion arranca solo si la respuesta de ingresar datos es = 1

`while(resp==1)`

El uso de auxciliares es para validarlos y luegos pasarles esos valores al struct

`cuentas[i].clave=auxClave;`

```
while(auxNroCuenta<=0){
            cout<<endl<<"El numero de cuenta tiene que ser mayor a 0,
vuelva a ingresarlo ";
            cin>>auxNroCuenta;
        }
        cuentas[i].nroCuenta=auxNroCuenta;
//Valida que los nros de cuenta ingresados sean mayores que 0

//Una vez validado el valor es paso a la posicion del arreglo del struct
```

```
        cout<<endl<<"Ingrese el saldo incial de la cuenta: ";
        cin>>cuentas[i].saldoIncial;

        cuentas[i].saldoActual=cuentas[i].saldoIncial;
//Pide informar el saldo inicial de la cuenta por el usuario

//Luego ese valor es guardado en el saldo actual, para ser modificado
constantemente
```

```
        cout<<"Hay mas cuentas para cargar?(Si=1/No=0) "<<endl;
        cin>>resp;

        while((resp!=0)&&(resp!=1)){

            cout<<"Respuesta mal ingresada(Si=1/No=0), por favor vuelva a
ingresarla"<<endl;
            cin>>resp;
        }
//Vuelve a preguntar para salir del bucle
```

```
    cuentas[i].contExtraccion=0;
    //Se inicializa el contador de extracciones para esta cuenta en 0, ya
que es una cuenta nueva.
        i++;
        //Incrementa el contador i para seguir cargando la próxima cuenta.
    }

    dim=i;
    //Asigna el valor de i (el número de cuentas cargadas) a la variable
dim por referencia, de modo que fuera de la función, se conozca cuántas
cuentas se cargaron en total.
```

Funcion para validar que los digitos de la cuenta sean 4

```
void claveDigitos(int &auxClave){

    while((auxClave>9999)||(auxClave<1000)){

        cout<<"La clave debe tener 4 Digitos, ingresela nuevamente: ";
        cin>>auxClave;
    }

}
```

- Se utiliza para validar una clave de seguridad de 4 dígitos
- void `claveDigitos(int &auxClave)`: Esto define la función llamada claveDigitos, que toma un parámetro por referencia llamado auxClave.
    - El uso de una referencia (&) significa que los cambios hechos en auxClave dentro de esta función afectarán a la variable original desde donde se llamó la función.
- El cuerpo de la función contiene un bucle while. Este bucle se ejecutará mientras la condición entre paréntesis sea verdadera.
    - while((auxClave > 9999) || (auxClave < 1000)): La condición de este bucle verifica si auxClave es mayor que 9999 o menor que 1000. En otras palabras, se asegura de que auxClave tenga exactamente 4 dígitos.

Funcion para imprimir los datos que les pasen

```
void imprimir(Datos vec[],int n){
    int i;
    for(i=0;i<n;i++){

cout<<endl<<"*****************************************************"<<endl;
        cout<<vec[i].nroCuenta<<endl;
        cout<<vec[i].limiteExtraccion<<endl;
        cout<<vec[i].saldoIncial<<endl;
        cout<<vec[i].saldoActual<<endl;
        cout<<vec[i].clave<<endl;
    }
}
```

- Se utiliza para mostrar en la consola la información de un arreglo de estructuras de tipo Datos.
- El cuerpo de la función contiene un bucle for, que se utiliza para recorrer el arreglo y mostrar los datos de cada estructura Datos

Funcion para calcular el promedio de todos los saldos actuales

```
double promedio(Datos vec[], int n){
    int i, acum=0;
    double promedio;

    for(i=0;i<n;i++){
        acum+=vec[i].saldoActual;
    }
    promedio=acum/n;

    return promedio;

}
```

- `double promedio(Datos vec[], int n)`: Esto define la función promedio. Toma dos argumentos: un arreglo de estructuras Datos llamado `vec` y un entero `n`, que representa el número de elementos en el arreglo.
- Se declaran dos variables locales: i, que se usará para iterar a través del arreglo, y acum, que se utilizará para acumular la suma de los saldos actuales.
- `for (i = 0; i < n; i++)`: Esto inicia un bucle for que recorre el arreglo de cuentas, comenzando desde `i = 0` y continuando hasta `i` sea menor que `n` (el número de elementos en el arreglo).
    - `acum += vec[i].saldoActual;`: En cada iteración del bucle, se agrega el valor del saldo actual de la cuenta en la posición `i` al acumulador `acum`. Esto suma todos los saldos actuales de las cuentas en el arreglo.
- Después de completar el bucle, se calcula el promedio dividiendo la suma acumulada (acum) entre el número de cuentas (n).
- return promedio;: La función devuelve el valor del promedio calculado.

Funcion para validar el nro de la cuenta

```
int validarNRoCuenta(Datos vec[],int n,int resp){
    int i,m;
    i=0;
    while((i<n)&&(vec[i].nroCuenta!=resp)&&(resp!=0)){

        i++;
    }

    if(i==n){
        m=-1;
    }
    else{
        m=i;
    }

    return m;

}
```

- Busca un número de cuenta específico dentro de un arreglo de estructuras Datos.
- Se declaran dos variables locales: `i`, que se utilizará para iterar a través del arreglo, y `m`, que se usará para almacenar el resultado de la búsqueda.
- Bucle `while` se inicia, y su condición es que `i` sea menor que `n`, lo que garantiza que no se desborde el arreglo y que se busque en todos los elementos del arreglo. Además, se verifica que el número de cuenta de la estructura en la posición i no sea igual a `resp` y que `resp` no sea igual a 0. Si se cumple cualquiera de estas condiciones, el bucle continúa ejecutándose.
- Dentro del bucle, i se incrementa en cada iteración, lo que permite buscar en la siguiente posición del arreglo. `i++;`
- Después de completar el bucle, se verifica si i es igual a n. Si es cierto, significa que no se encontró el número de cuenta deseado en el arreglo, y en ese caso, se establece m en -1 para indicar que no se encontró el número de cuenta.

```
if(i==n){
        m=-1;
    }
    else{
        m=i;
    }
```

- Si `i` no es igual a `n, entonces se ha encontrado el número de cuenta en una posición válida del arreglo, y m se establece en el valor de` i` para indicar la posición en la que se encontró el número de cuenta.
- La función devuelve el valor de m, que será -1 si el número de cuenta no se encontró en el arreglo o el índice de la posición en la que se encontró si se encontró con éxito.

En resumen, esta función busca un número de cuenta específico dentro de un arreglo de estructuras Datos y devuelve su posición en el arreglo o -1 si no se encuentra

Funcion para validar la clave

```
bool validarClave(int n,int m){

    bool flag;

    if(n==m){
        flag=true;
    }
    else{
        flag=false;
    }

    return flag;
};
```

- Compara dos números enteros y devuelve un valor booleano true o false según si son iguales o no.
- `bool validarClave(int n, int m)`: Esto define la función validarClave. Toma dos argumentos, ambos son números enteros, `n` y m, que se van a comparar.
- Se declara una variable local flag de tipo bool que se usará para almacenar el resultado de la comparación.
- ;uego, se compara si n es igual a m utilizando el operador de igualdad ==. Si son iguales, se establece flag en true, lo que significa que la función devuelve true.
- Si n no es igual a m, se establece flag en false, lo que significa que la función devuelve false.
- La función devuelve el valor de flag, que será `true` si n y m son iguales y false si no lo son.

En resumen, esta función toma dos números enteros, compara si son iguales y devuelve `true` si lo son o false si no lo son. Se utiliza para validar si la clave ingresada coincide con la clave almacenada en la estructura Datos

Funcion para calcular las extracciones

```
void extrac(Datos vec[], int i, int e){
    if((vec[i].limiteExtraccion<e)){
        cout<<"la operacion no se pudo realizar, por que su monto de
extraccion diario se ah terminado"<<endl;
    }
    else{
        vec[i].limiteExtraccion-=e;
        vec[i].saldoActual-=e;
        cout<<"La operacion se realizo con exito, su monto actual es: $"
<<vec[i].saldoActual<<endl;//en el main
    }


}
```

- Se utiliza para realizar extracciones de dinero en una cuenta bancaria.
- `void extrac(Datos vec[], int i, int e)`: Esto define la función extrac. Toma tres argumentos:
    - vec[]: Un arreglo de estructuras Datos que representa las cuentas bancarias.
    - i: Un índice que indica la cuenta en la que se realizará la extracción.
    - e: El monto que se desea extraer de la cuenta.
- La función comienza con una condición if que verifica si el `limiteExtraccion` en la cuenta `vec[i]` es menor que e. Esto significa que si el monto solicitado para la extracción es mayor que el límite diario permitido, no se puede realizar la extracción.
- Si la condición del if es verdadera (es decir, el límite de extracción es menor que el monto deseado), la función imprime un mensaje que indica que la operación no se puede realizar debido a que el límite de extracción diario se ha agotado.
- Si la condición del if es falsa, significa que el monto solicitado para la extracción es igual o menor que el límite diario. En este caso, la función resta e del limiteExtraccion en la cuenta vec[i] y también resta e del saldoActual de la misma cuenta. Esto simula la extracción del monto de la cuenta.

Esta función se utiliza para gestionar las extracciones de dinero en el programa principal y realiza las validaciones necesarias antes de efectuar la operación.

Funcion para calcular el limite de extraciones

```
bool extraccionesLimite(Datos vec[],int i){
    bool flag;

    if(vec[i].contExtraccion<3){
        vec[i].contExtraccion++;
        flag=`true`;
    }
    if(vec[i].contExtraccion==3){
        flag=false;
    }

    return flag;
}
```

- Verifica si un usuario ha alcanzado el límite de extracciones diarias permitidas en una cuenta bancaria.
- `bool extraccionesLimite(Datos vec[], int i)`: Esto define la función extraccionesLimite. Toma dos argumentos:
  - `vec[]`: Un arreglo de estructuras Datos que representa las cuentas bancarias.
  - `i`: Un índice que indica la cuenta para la cual se verificará el límite de extracciones.
- La función comienza con una variable booleana `flag` que se utilizará para indicar si el usuario puede realizar otra extracción o si ha alcanzado el límite.
- La función `verifica si vec[i].contExtraccion` es menor que 3. `contExtraccion` es un contador que lleva el registro de la cantidad de extracciones realizadas en una cuenta. Si es menor que 3, significa que el usuario aún puede realizar más extracciones.
- Si la condición se cumple (el usuario no ha alcanzado el límite de extracciones), la función incrementa `vec[i].contExtraccion` en 1 y establece`flag` en `true`, lo que indica que la extracción se permite.
- Sin embargo, si `vec[i].contExtraccion` alcanza el valor de 3 (lo que significa que el usuario ha realizado 3 extracciones), se establece `flag` en `false`, lo que indica que el usuario ha superado el límite de extracciones diarias
- La función devuelve el valor de `flag`, que refleja si el usuario puede realizar otra extracción o no.

Esta función se utiliza para controlar el número de extracciones permitidas en una cuenta bancaria y asegurarse de que no se exceda el límite diario.

Funcion para el deposito

```
void depo(Datos vec[],int i,int d){
    vec[i].saldoActual+=d;
```

```
}
```

- Se utiliza para realizar un depósito en una cuenta bancaria
- `void depo(Datos vec[], int i, int d)`: Esto define la función depo. Toma tres argumentos:
  - `vec[]`: Un arreglo de estructuras Datos que representa las cuentas bancarias.
  - `i`: Un índice que indica la cuenta en la que se realizará el depósito.
  - `d: El monto que se va a depositar en la cuenta.
- La función simplemente suma el valor d al campo `vec[i].saldoActual`. El campo saldoActual representa el saldo actual de la cuenta en la posición i`` del arreglo `vec[]`.
- En otras palabras, esta función aumenta el saldo actual de la cuenta en la que se realiza el depósito en la cantidad especificada por d.

En resumen, la función depo se utiliza para aumentar el saldo actual de una cuenta bancaria cuando se realiza un depósito en esa cuenta.

Funcion para calcular el tipo de cuentas

```
void cantidadCuentas(Datos vec[],int n,int &contAcreedor, int &contDeudor,
int &contNulo){
    int i;

    for(i=0;i<n;i++){
        if(vec[i].saldoActual>=1){
            contAcreedor++;
        }
        else{
            if(vec[i].saldoActual<=-1){
                contDeudor++;
            }
            else{
                contNulo++;
            }
        }
    }
}
```

- Se utiliza para contar y clasificar las cuentas bancarias en tres categorías: cuentas acreedoras, cuentas deudoras y cuentas nulas.
- `void cantidadCuentas(Datos vec[], int n, int &contAcreedor, int &contDeudor, int &contNulo)`: Esto define la función cantidadCuentas. Toma cinco argumentos:
  - `vec[]`: Un arreglo de estructuras Datos que representa las cuentas bancarias.
  - `n`: El número de cuentas en el arreglo.
  - `contAcreedor`: Una referencia a una variable que almacenará la cantidad de cuentas acreedoras.
  - `contDeudor`: Una referencia a una variable que almacenará la cantidad de cuentas deudoras.

    - `contNulo`: Una referencia a una variable que almacenará la cantidad de cuentas nulas.
- La función recorre un bucle `for` que itera a través de todas las cuentas en el arreglo `vec[]`.
- En cada iteración del bucle, se verifica el saldo actual de la cuenta en la posición i del arreglo:
    - Si `vec[i].saldoActual` es mayor o igual a 1, se incrementa el contador de `contAcreedor`.
    - Si `vec[i].saldoActual` es menor o igual a -1, se incrementa el contador de `contDeudor`.
    - En cualquier otro caso (saldo igual a 0), se incrementa el contador de `contNulo`.
- Al final de la función, `contAcreedor`contendrá la cantidad de cuentas con saldos mayores o iguales a 1, `contDeudor` contendrá la cantidad de cuentas con saldos menores o iguales a -1, y `contNulo` contendrá la cantidad de cuentas con saldos iguales a 0.

Esta función se utiliza para clasificar las cuentas y contar cuántas están en cada categoría con el propósito de generar estadísticas sobre el estado de las cuentas bancarias.