Universidad Nacional de la Matanza

Elementos de Programación

UNIDAD 11. CORTE DE CONTROL

INDICE

UNIDAD 11
CORTE DE CONTROL

OBJETIVOS: Procesar lotes de datos ordenados a fin de obtener reportes o estadísticas. Afianzar el manejo de estructuras repetitivas, contadores y acumuladores.

1. ¿Qué es el corte de control?

El corte de control es un proceso en el cual partiendo de registros ordenados por el valor de uno o más campos se los procesa para obtener información agrupada en lotes y sub-lotes al detectar un cambio en la/las variables por la cual la información se encuentra organizada.

En otras palabras, se procesa un conjunto ordenado de registros en subconjuntos determinados por el cambio de valor de una o más variables.

El corte de control se aplica para obtener información detallada agrupada por uno o más criterios.

Dicha información puede mostrarse en forma de reporte a medida que se detecta el cambio en la variable por la cual la información se encuentra ordenada o puede grabarse un archivo resumen con dicha información.

La condición necesaria para aplicar corte de control es que la información esté ordenada u organizada por el campo sobre el cual queremos agrupar la información, es decir que todos los registros de un mismo tipo deben estar juntos.

El corte de control puede aplicarse en varios niveles siempre y cuando la información venga ordenada por todos los campos sobre los cuales se desea agrupar. Por ejemplo, si se desea obtener información de los ingresos a un establecimiento agrupada por día y en cada día se desea obtener un detalle por hora, los datos deben estar primero ordenados por día y dentro de cada día ordenados por hora. En este caso podría aplicarse dos cortes uno por día y otro por hora a fin de obtener información agrupada en cada uno de ellos. Pero no siempre que la información esté ordenada significa que se deba aplicar corte de control. En el caso anterior si solo se desea obtener información detallada por día, entonces se hará un corte de un solo nivel ya que agrupar también por hora no aportaría información adicional.

Para aplicar corte de control si la información se ingresa por teclado debe ingresarse en forma ordenada, es por ello por lo que lo más común es procesar información ya ordenada que venga en un vector o en un archivo.

2. Reglas Generales para aplicar corte de control

El corte de control se realiza anidando distintas estructuras repetitivas con las siguientes reglas:

- Se realiza un ciclo repetitivo con la condición de fin de datos
- Dentro del ciclo de fin de datos se anida un ciclo por cada nivel de corte que se desea realizar (el ciclo de cada nivel va anidado dentro del ciclo del nivel anterior)
- Las condiciones de los ciclos se van propagando, es decir que el ciclo interno de corte va a incluir la condición de fin del ciclo que lo contiene y su propia condición. Si hay otros niveles de corte va a incluir las condiciones anteriores y su propia condición de fin.
- La información se lee antes de ingresar al ciclo de la condición de fin y se vuelve a leer antes de finalizar el ciclo más interno.

- Las variables que cuentan o acumulan información agrupada deben inicializar antes de comenzar el ciclo del corte y mostrarse al salir de dicho ciclo haciendo referencia a que dicha información corresponde al dato leído anteriormente y no al último, ya que el ciclo termina por que el lote finalizó y por lo tanto la lectura actual va a ser diferente.

3. Cómo aplicar Corte de Control en diferentes casos.

3.1 Un nivel de corte con ingreso desde teclado

Supongamos que en el Departamento de Alumnos de la UNLaM se quiere saber cuántos alumnos se inscribieron en cada Departamento como así también la cantidad total de alumnos inscriptos en la Universidad, considerando que los datos están ordenados por el campo clave "Número de Departamento Inscripto", para lo cual se han analizado los siguientes datos:

Estructura **PERSONA**				
DNI	Apellido y Nombre	Nro Dpto Inscripto	Nro. Comisión	Código de Materia

Ejemplo de la Lista de Datos a analizar:

nDNI	ApyN	nDpto	nComi	cMat	
46051476	Martínez, Lucia	1	2	1046	
45765812	Franco, Luciano	1	2	1230	
46256892	Ramírez, María	1	3	1501	
6785933	Luna, Ignacio	1	3	1530	
45368947	López, Roxana	1	5	1501	
46789256	Mangano, Luis	1	6	1501	6
44589632	Lu, Lian	2	1	1680	
45635484	Marengo, Nicolás	2	1	1695	
46489212	Pellejero, Diana	2	2	1680	
45632584	Herrera, Mario	2	4	1623	
46869572	Amaya, Luciara	2	4	1625	
46759154	Pedrazza, Antonio	2	6	1680	6
45389456	Ramírez, Mario	4	1	1742	
46786428	Murua, Analía	4	1	1748	
46348922	Moreno, Claudio	4	1	1748	
46555789	Velloso, Andrea	4	5	1723	4
46893768	Luciani, Marilu	5	5	1944	
45785354	Perrota, Mario	5	5	1945	
46223566	Landaburo, Lucila	5	6	1923	
45883698	Randazzo, Emiliano	5	6	1924	4
99999999					
					20

De acuerdo al requerimiento formulado el resultado que se obtendrá del anterior juego de datos será:

Departamento	Cantidad de Alumnos
1	6
2	6
4	4
5	4

Total de alumnos en la Universidad: 20

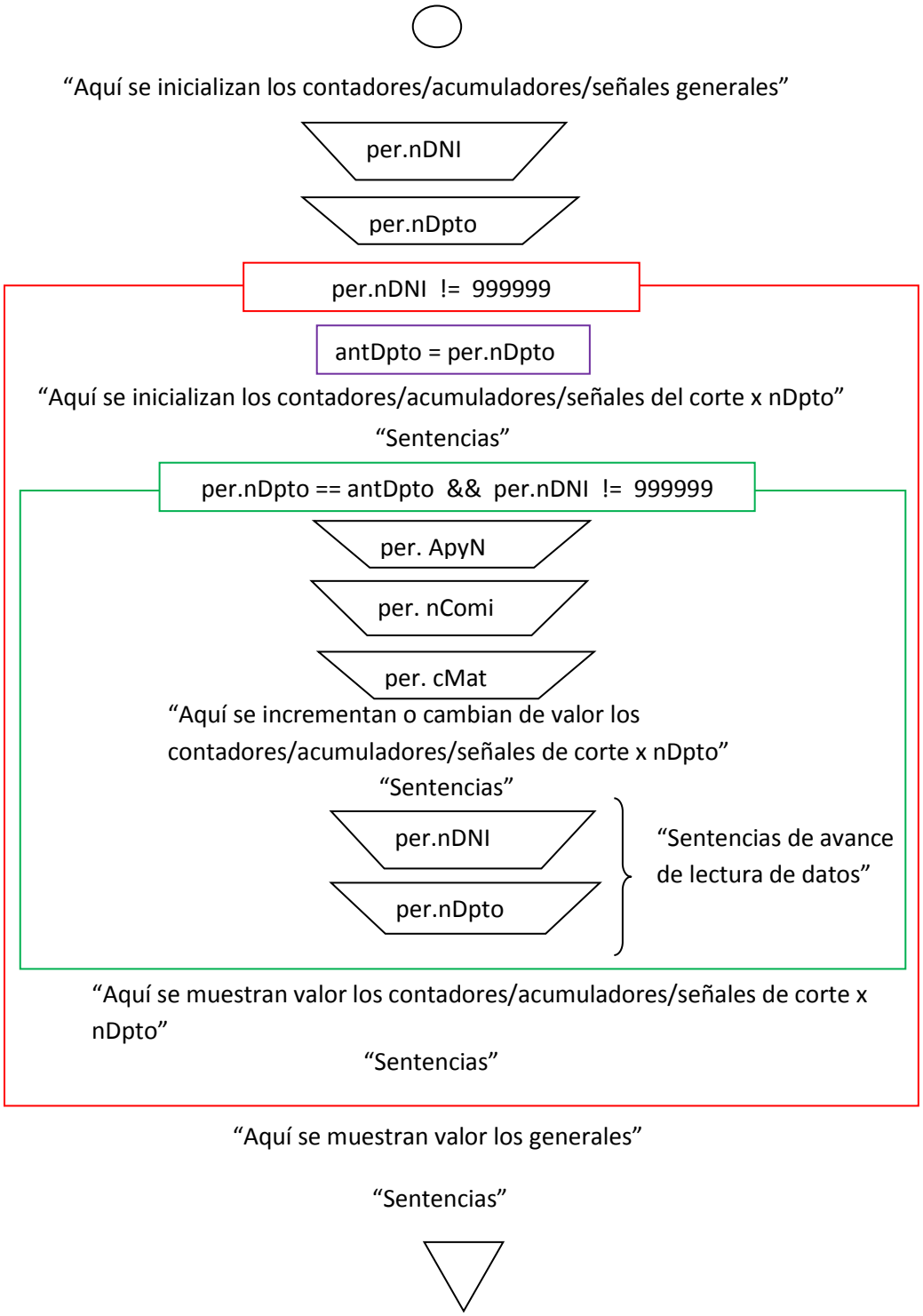
Estrategia para resolver el problema:

Se desea mostrar información agrupada por departamento por lo tanto los datos se deben ingresar ordenados por número de departamento. Por cada ingreso se debe recordar cuál fue el número de departamento que se ingresó anteriormente y compararlo con el recién ingresado. Si el nuevo dato es distinto del anterior entonces se detecta que se finalizó de ingresar la información del lote anterior. Por cada lote se debe contar cuantos inscriptos hay entonces se debe utilizar un contador que debe inicializarse cada vez que se comienza un nuevo lote de información y mostrarse al detectar que el mismo finaliza.

Diagrama de Lógica general para el análisis:

Variables de la función main ()

```
struct  PERSONA  per;
int antDpto;
```


"Aquí se inicializan los contadores/acumuladores/señales generales"
per.nDNI
per.nDpto
per.nDNI != 999999
antDpto = per.nDpto
"Aquí se inicializan los contadores/acumuladores/señales del corte x nDpto"
"Sentencias"
per.nDpto == antDpto && per.nDNI != 999999
per. ApyN
per. nComi
per. cMat
"Aquí se incrementan o cambian de valor los contadores/acumuladores/señales de corte x nDpto"
"Sentencias"
per.nDNI
per.nDpto
"Sentencias de avance de lectura de datos"
"Aquí se muestran valor los contadores/acumuladores/señales de corte x nDpto"
"Sentencias"
"Aquí se muestran valor los generales"
"Sentencias"

Análisis de Diagrama de Lógica:

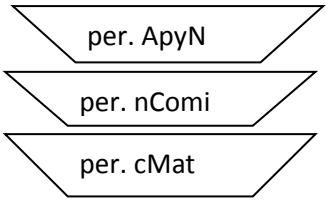
- Hay 2 **while** el primero o sea el externo siempre será la “Condición de Fin” y el segundo o sea el interno será el “Corte de Control”.
- Las variables que deben estar siempre antes del primer/externo **while** son las que están dentro de las “condiciones de cada uno de los **while**”.
- Siempre antes del **while** que será el “Corte de Control” deberá estar la asignación a una variable auxiliar que le permitirá guardar el contenido del “campo clave”, y de esta manera controlar cuando se realiza al cambio del contenido de la variable “campo clave”. En este ejemplo será:

```
antDpto = per.nDpto
```

- En cada **while** se arrastra la condición del **while** anterior y se le agrega con un operador lógico **AND** la condición del nuevo Corte de Control. En este ejemplo será:

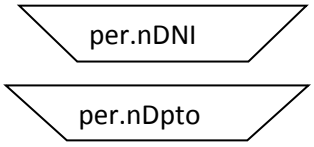
```
per.nDpto == antDpto  &&  per.nDNI  !=  999999
```

- Dentro del **while** mas interno deben estar el resto de las sentencias que corresponden a



todas las variables que forman parte de los datos, en este ejemplo serán:

- El incremento de la lectura se realiza dentro del **while** más interno. Allí es donde se avanza con la lectura de datos, que en este ejemplo serán:



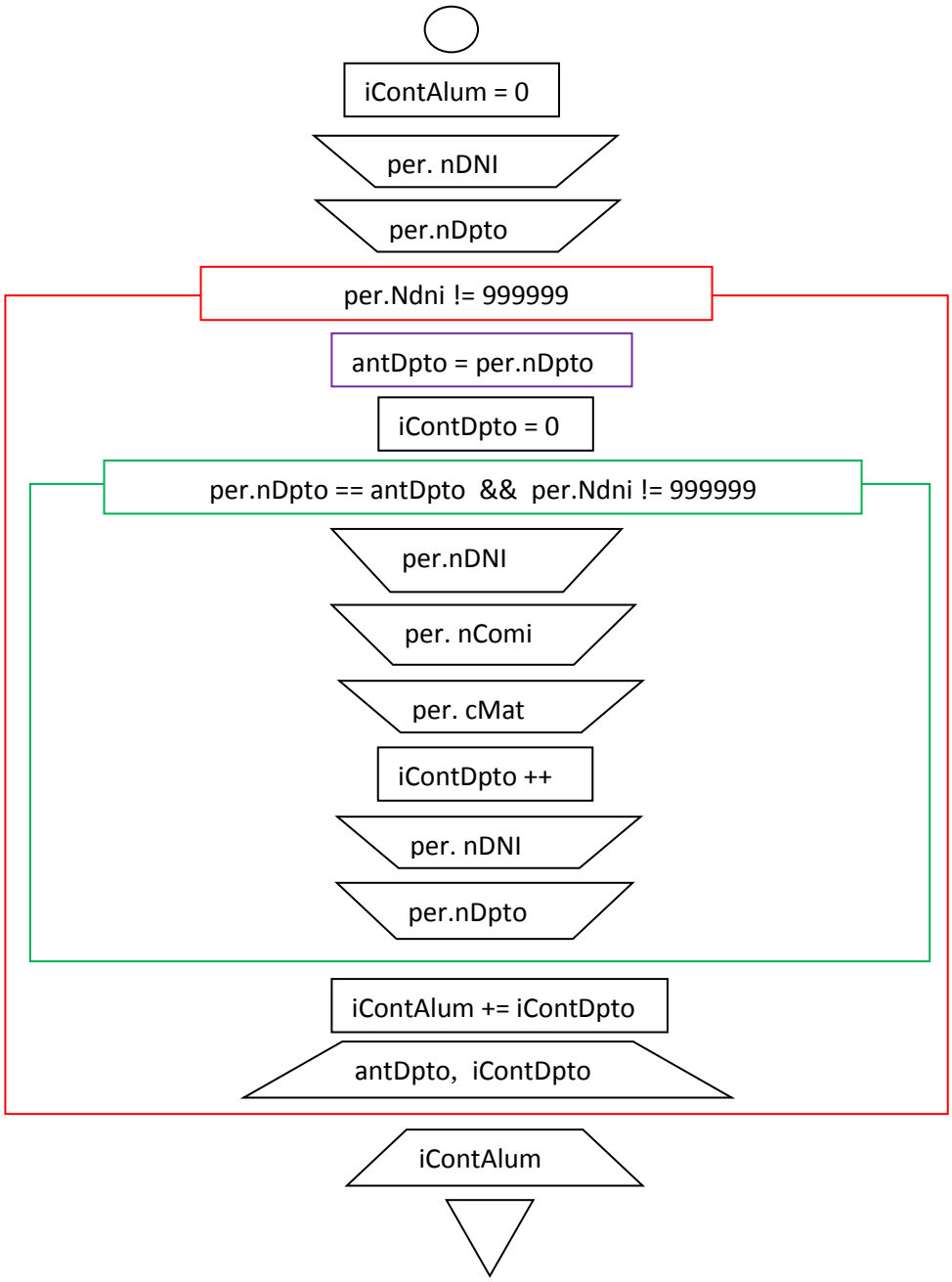


Diagrama completo:

Codificación en Lenguaje C:

```
#include <conio.h>
#include <stdio.h>
#include <string.h>

struct PERSONA
{
	int nDNI;
	char ApyN [26];
	int nDpto;
	int nComi;
	int cMat;
};
```

```
int main (  )
{
    struct PERSONA per;
    int antDpto, iContAlum, iContDpto;

    iContAlum = 0;

    printf("\nIngrese Numero de DNI (999999 para terminar): ");
    scanf("%d",&per.nDNI);

    printf("\nIngrese Numero de Departamento: ");
    scanf("%d",&per.nDpto);

    while (per.nDNI != 999999)
    {
        antDpto = per.nDpto;
        iContDpto = 0;

        while (per.nDpto == antDpto && per.nDNI != 999999)
        {
            getchar(); //limpia el buffer
            printf("\nIngrese Apellido y Nombre: ");
            gets(per.ApyN);

            printf("\nIngrese Numero de Comision: ");
            scanf("%d",&per.nComi);

            printf("\nIngrese Codigo de Materia: ");
            scanf("%d",&per.cMat);

            iContDpto ++;

            printf("\nIngrese Numero de DNI (999999 para terminar): ");
            scanf("%d",&per.nDNI);

            printf("\nIngrese Numero de Departamento: ");
            scanf("%d",&per.nDpto);
        }

        iContAlum += iContDpto;
        printf("\nLa Cantidad de Alumnos de Nro Departamento %d es %d", antDpto,
iContDpto);
    }

   printf("\nLa Cantidad de Alumnos que tiene la Universidad %d", iContAlum);
   return 0;
}
```

3.2 Dos Niveles de corte con ingreso desde teclado

Supongamos que en el Departamento de Alumnos de la UNLaM se quiere saber cuántos alumnos se inscribieron en cada Comisión y en cada Departamento como así también la cantidad total de alumnos inscriptos en la Universidad, considerando que los datos están ordenados por el campo clave “Número de Departamento Inscripto” y además por “Número de Comisión”, para lo cual se han analizado los siguientes datos:

Ejemplo de la Estructura de Datos:

Estructura **PERSONA**				
DNI	Apellido y Nombre	Nro Dpto Inscripto	Nro. Comisión	Código de Materia

Ejemplo de la Lista de Datos a analizar:

nDNI	ApyN	nDpto	nComi	cMat	
46051476	Martínez, Lucia	1	2	1046	
45765812	Franco, Luciano	1	2	1230	2
46256892	Ramírez, María	1	3	1501	
46785933	Luna, Ignacio	1	3	1530	2
45368947	López, Roxana	1	5	1501	
46789256	Mangano, Luis	1	6	1501	
44589632	Lu, Lian	2	1	1680	
45635484	Marengo, Nicolás	2	1	1695	
46489212	Pellejero, Diana	2	2	1680	
45632584	Herrera, Mario	2	4	1623	
46869572	Amaya, Luciara	2	4	1625	
46759154	Pedrazza, Antonio	2	6	1680	6
45389456	Ramírez, Mario	4	1	1742	
46786428	Murua, Analía	4	1	1748	
46348922	Moreno, Claudio	4	1	1748	
46555789	Velloso, Andrea	4	5	1723	4
46893768	Luciani, Marilu	5	5	1944	
45785354	Perrota, Mario	5	5	1945	
46223566	Landaburo, Lucila	5	6	1923	
45883698	Randazzo, Emiliano	5	6	1924	4
99999999					
					20

De acuerdo al requerimiento formulado el resultado que se obtendrá del anterior juego de datos será:

Departamento	Comisión	Cantidad de Alumnos
1	2	2
1	3	2
1	5	1
1	6	1
1		6
2	1	2
2	2	1
2	4	2
2	6	1
2		6

4	1	3
4	5	1
4		4
5	5	2
5	6	2
5		4

Total de alumnos en la Universidad: 20

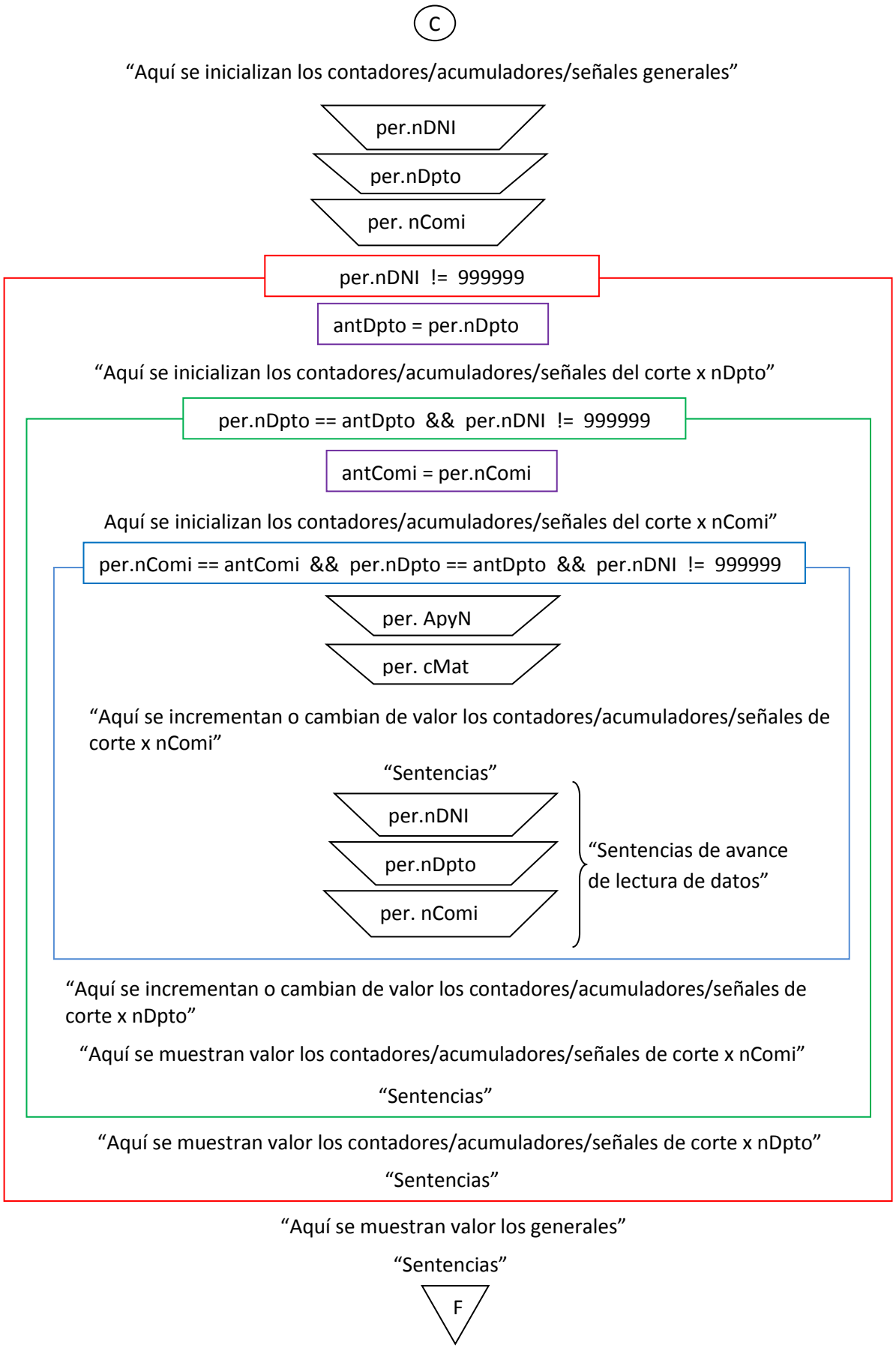
Estrategia para resolver el problema:

Ahora lo que se pide es la información agrupada por departamento y dentro de cada departamento agrupados por comisión, por lo tanto, serán necesarios dos niveles de corte. Los datos se deben ingresar ordenados por departamento y comisión para poder ser procesados correctamente. Vamos a necesitar 3 ciclos, uno con la condición de fin, uno para el corte por departamento y otro para el corte por comisión.

Diagrama de Lógica para el análisis:

Variables de la función main():

```
struct PERSONA per;
int antDpto, antComi;
```


C
"Aquí se inicializan los contadores/acumuladores/señales generales"
per.nDNI
per.nDpto
per. nComi
per.nDNI != 999999
antDpto = per.nDpto
"Aquí se inicializan los contadores/acumuladores/señales del corte x nDpto"
per.nDpto == antDpto && per.nDNI != 999999
antComi = per.nComi
Aquí se inicializan los contadores/acumuladores/señales del corte x nComi"
per.nComi == antComi && per.nDpto == antDpto && per.nDNI != 999999
per. ApyN
per. cMat
"Aquí se incrementan o cambian de valor los contadores/acumuladores/señales de corte x nComi"
"Sentencias"
per.nDNI
per.nDpto
per. nComi
"Sentencias de avance de lectura de datos"
"Aquí se incrementan o cambian de valor los contadores/acumuladores/señales de corte x nDpto"
"Aquí se muestran valor los contadores/acumuladores/señales de corte x nComi"
"Sentencias"
"Aquí se muestran valor los contadores/acumuladores/señales de corte x nDpto"
"Sentencias"
"Aquí se muestran valor los generales"
"Sentencias"
F

Análisis de Diagrama de Lógica:

- Hay 3 **while** el primero o sea el externo siempre será la "Condición de Fin", el segundo y el tercero serán los "Cortes de Control".
- Las variables que deben estar siempre antes del primer/externo **while** son las que están dentro de las "condiciones de cada uno de los **while**".
- **Siempre** antes de cada **while** que será el "Corte de Control" deberá estar la asignación a una variable auxiliar que le permitirá guardar el contenido del "campo clave", y de esta manera controlar cuando se realiza al cambio del contenido de la variable "campo clave". En este ejemplo será:

  antDpto = per.nDpto

  antComi = per.nComi

- En cada **while** se arrastra la condición del **while** anterior y se le agrega con un operador lógico **AND** la condición del nuevo Corte de Control. En este ejemplo será:

  per.nDpto == antDpto && per.nDNI != 999999

  per.nComi == antComi && per.nDpto == antDpto && per.nDNI != 999999

- Dentro del **while** mas interno deben estar como las primeras sentencias el resto de todas las variables que forman parte de los datos, en este ejemplo serán:

  per. ApyN

  per. cMat

- El incremento de la lectura se realiza dentro del **while** más interno. Allí es donde se avanza con la lectura de datos, que en este ejemplo serán:

  per.nDNI

  per.nDpto

  per. nComi

3.3 Un Nivel de corte desde un archivo

En la Empresa “M. y M.” se quiere saber cuánto se abona de sueldo por cada Sector de Fábrica y en toda la Empresa. Los datos están ordenados por el campo clave “Número de Sector de Fábrica”, para lo cual se han analizado los datos que se encuentra en el archivo llamado “empleados.dat” y los registros del archivo tiene el siguiente formato:

a. Número de Legajo (entero)

b. Apellido y Nombre (hasta un máximo de 25 letras)

c. Año de Ingreso a la empresa (entero)

d. Número de Sector de Fábrica (entero)

e. Sueldo (real)

Resolución del problema:

La propuesta es leer y no bajar a memoria los datos que tiene el archivo llamado “empleados.dat” porque se desconoce la cantidad de Registros que posee el archivo, y para lo cual se necesita crear una Estructura que contenga el formato del Registro del archivo, con lo cual se propone crear una estructura que se llamará PERSONA, que será la siguiente:

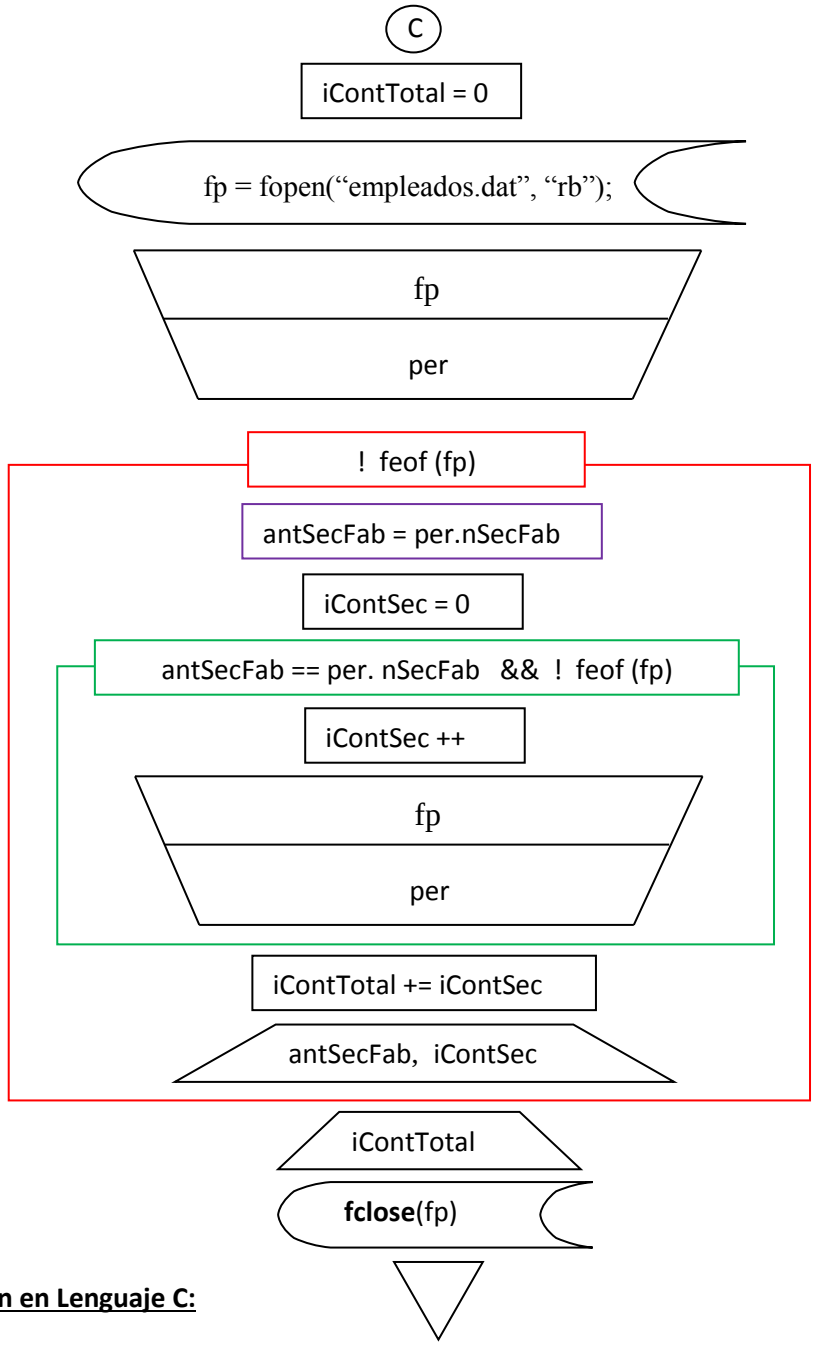
Estructura **PERSONA**				
Legajo	Apellido y Nombre	Año Ingreso	Nro.Sector Fábrica	Sueldo

Diagrama de Lógica:

struct PERSONA				
int nLegajo	**char** ApyN [26]	**int** nAnio	**int** nSecFab	**float** rSueldo

Variables de la función main():

```
struct  PERSONA  per;
FILE  fp
int antSecFab,iContTotal, iContSec;
```



Codificación en Lenguaje C:

```
#include <conio.h>
#include <stdio.h>
#include <string.h>

struct PERSONA
{
      int nLegajo;
      char ApyN [26];
      int nAnio;
      int nSecFab;
      float rSueldo;
};

int main (  )
{
      struct PERSONA per;
      int antSecFab, iContTotal, iContSec;
```

```
	FILE *fp;
	iContTotal = 0;
	if ( (fp = fopen("empleado.dat", "rb")) == NULL )
	{
		printf("\nNo se puede abrir.");
		getch();
		exit(1);
	}

	fread(&per, sizeof(struct PERSONA ),1,fp);
	while (! feof(fp))
	{
		antSecFab = per.nSecFab;
		iContSec = 0;

		while (per.nSecFab == antSecFab && ! feof(fp) )
		{

			iContSec ++;
			fread(&per, sizeof(struct PERSONA ),1,fp);

		}
		iContTotal += iContSec;
		printf("\nLa Cantidad de Empleados del Sector %d es %d",
		antSecFab, iContSec);
	}
	fclose(fp);

	printf("\nLa Cantidad de Empleados que tiene la empresa es %d",
	iContTotal);

	return 0;
}
```

3.4 Un Nivel de corte desde un archivo generando archivo resumen

Se dispone de un archivo llamado “ventas.dat” que tiene las ventas realizadas por una empresa a lo largo del mes. La empresa tiene distintas sucursales a lo largo del país. El archivo contiene registros con los siguientes datos:

- Codigo de sucursal (entero)
- Codigo de producto (entero)
- Cantidad Vendida (entero)
- Precio Unitario (real)

Para una misma sucursal se recibe un solo registro por cada producto vendido.

Se desea realizar un programa que calcule las ventas realizadas en cada sucursal generando un archivo llamado ventasXsuc.dat que contenga un registro sumarizado por cada sucursal con los siguientes datos:

- Codigo de sucursal (entero)
- Cantidad total de productos vendidos
- Importe total recaudado
- Codigo de producto con mayor cantidad de unidades vendidas (considerar único)

Resolución del problema:

Para resolver este problema será necesario trabajar con los dos archivos abiertos, el archivo ventas.dat se abrirá en modo lectura para ir leyendo registro a registro y hacer el corte de control sobre el campo código de sucursal. Al mismo tiempo se debe crear el archivo ventasXsuc.dat para que cada vez que se detecte un cambio en el código de sucursal se grabé un archivo sumarizado en dicho archivo.

Además de dos campos sumarizados (cantidad e importe), se nos pide determinar el código de producto con mayor cantidad de unidades vendidas por lo que se deberá aplicar el algoritmo para calcular el máximo para los productos vendidos en cada sucursal.

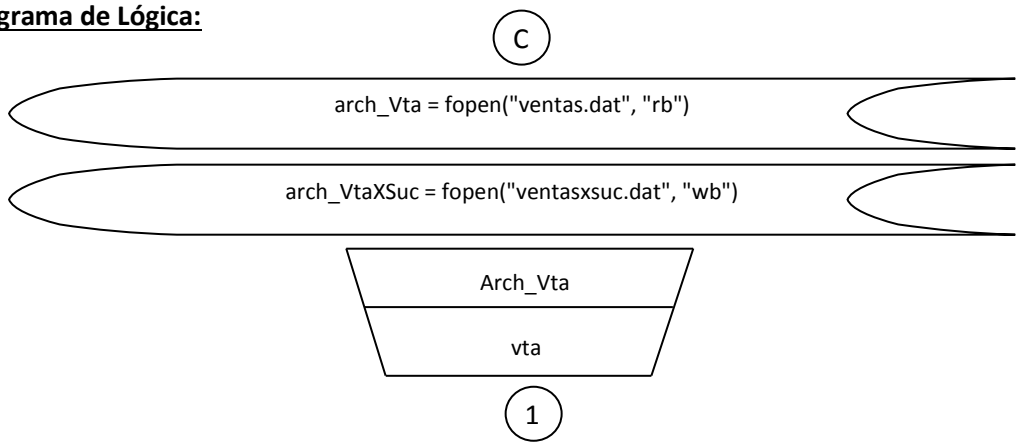
Se deben declarar dos estructuras de datos una por cada archivo ya que almacenan registros distintos.

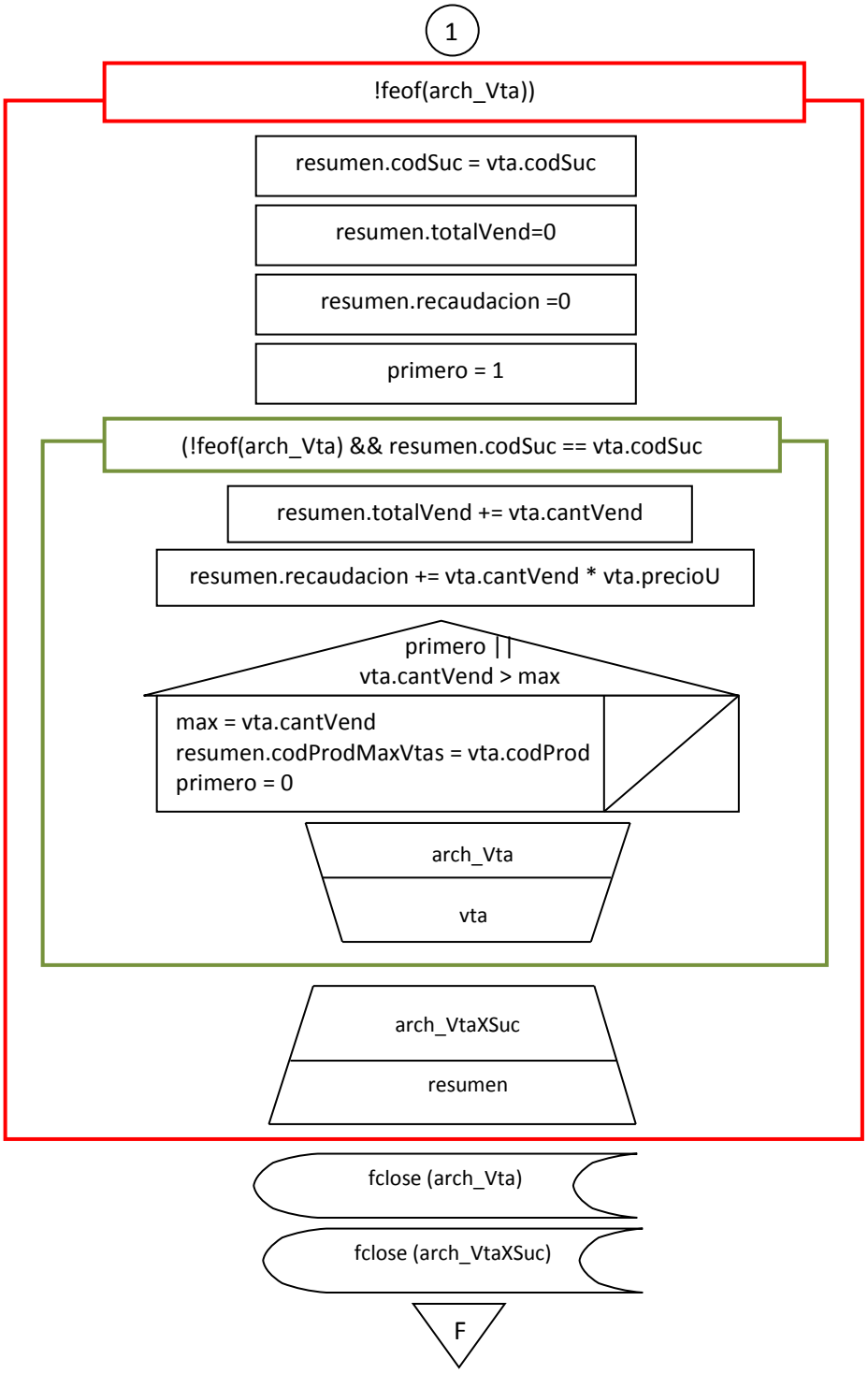
Estructura **sVenta**			
codSuc	codProd	cantVend	precioU

Estructura **sResumen**			
codSuc	totalVend	recaudacion	codProdMaxVtas

Nótese que este programa no tiene salida por pantalla los resulados se guardan directamente en un arhivo.

Diagrama de Lógica:





Codificación en Lenguaje C:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

struct sVenta
{
    int codSuc;
    int codProd;
    int cantVend;
    float precioU;
};
```

```
struct sResumen
{
    int codSuc;
    int totalVend;
    float recaudacion;
    int codProdMaxVtas;
};

int main()
{
    FILE * arch_Vta, * arch_VtaXSuc;
    struct sVenta vta;
    struct sResumen resumen;
    int primero, max;

    arch_Vta = fopen("ventas.dat", "rb");
    arch_VtaXSuc = fopen("ventasxsuc.dat", "wb");
    if (arch_Vta==NULL || arch_VtaXSuc==NULL)
    {
        printf("Error al abrir los archivos");
        getch();
        exit(1);
    }

    fread(&vta, sizeof(vta), 1, arch_Vta);
    while (!feof(arch_Vta))
    {
        /*guarda la sucursal anterior e inicializa los acumuladores
        y contadores directamente en la variable del tipo estructura
        que se va a grabar el archivo por sucursal*/

        resumen.codSuc = vta.codSuc;
        resumen.totalVend=0;
        resumen.recaudacion =0;
        primero = 1; //para el maximo

        while (!feof(arch_Vta) && resumen.codSuc == vta.codSuc)
        {
            resumen.totalVend += vta.cantVend;
            resumen.recaudacion += vta.cantVend * vta.precioU;

            if(primero || vta.cantVend > max)
            {
                max = vta.cantVend;
                resumen.codProdMaxVtas = vta.codProd;
                primero = 0;
            }
            fread(&vta, sizeof(vta), 1, arch_Vta);
        }
        fwrite (&resumen, sizeof( resumen),1, arch_VtaXSuc);
    }
    fclose(arch_Vta);
    fclose(arch_VtaXSuc);
}
```

3.5 Un Nivel de corte desde un archivo generando archivos dinámicamente

El sistema de control de ingreso y salida de empleados de una fábrica deja al final del mes un archivo ordenado por sector y sumarizado por empleado llamado horasTrabajadas.dat con los siguientes datos:

- Sector (texto de 10 caracteres máximo)
- DNI del empleado (entero)
- Horas trabajadas

Por otro lado se dispone del archivo empleados.dat con los empleados de la fábrica que contiene los siguientes datos:

- DNI del empleado (entero)
- Nombre y Apellido (texto de 30 caracteres máximo)
- Valor que cobra por hora (float)

No se sabe la cantidad exacta de empleados pero sí se sabe que no son más de 100.
Se desea realizar un programa que utilizando ambos archivos genere para cada sector un archivo que contenga el detalle de sueldos que deba pagar a sus empleados. Se debe generar para cada sector un archivo distinto cuyo nombre sea nombreSector.dat y debe contener registros con la siguiente información:

- DNI del empleado (entero)
- Nombre y Apellido (texto de 30 caracteres máximo)
- Horas trabajadas (entero)
- Sueldo a Pagar (float)

Resolución del problema:

Inicialmente se deben descargar en memoria los datos los empleados de la empresa, que como se sabe que no son más de 100 se pueden descargar en un vector.

Luego se procesa el archivo horasTrabajadas.dat que está ordenado por sector. Por cada sector se debe generar dinámicamente un nuevo archivo con el nombre dicho sector y calcular en él la información solicitada pare ello se debe buscar el empleado en el vector en memoria para obtener su nombre y valor que cobrar por hora.

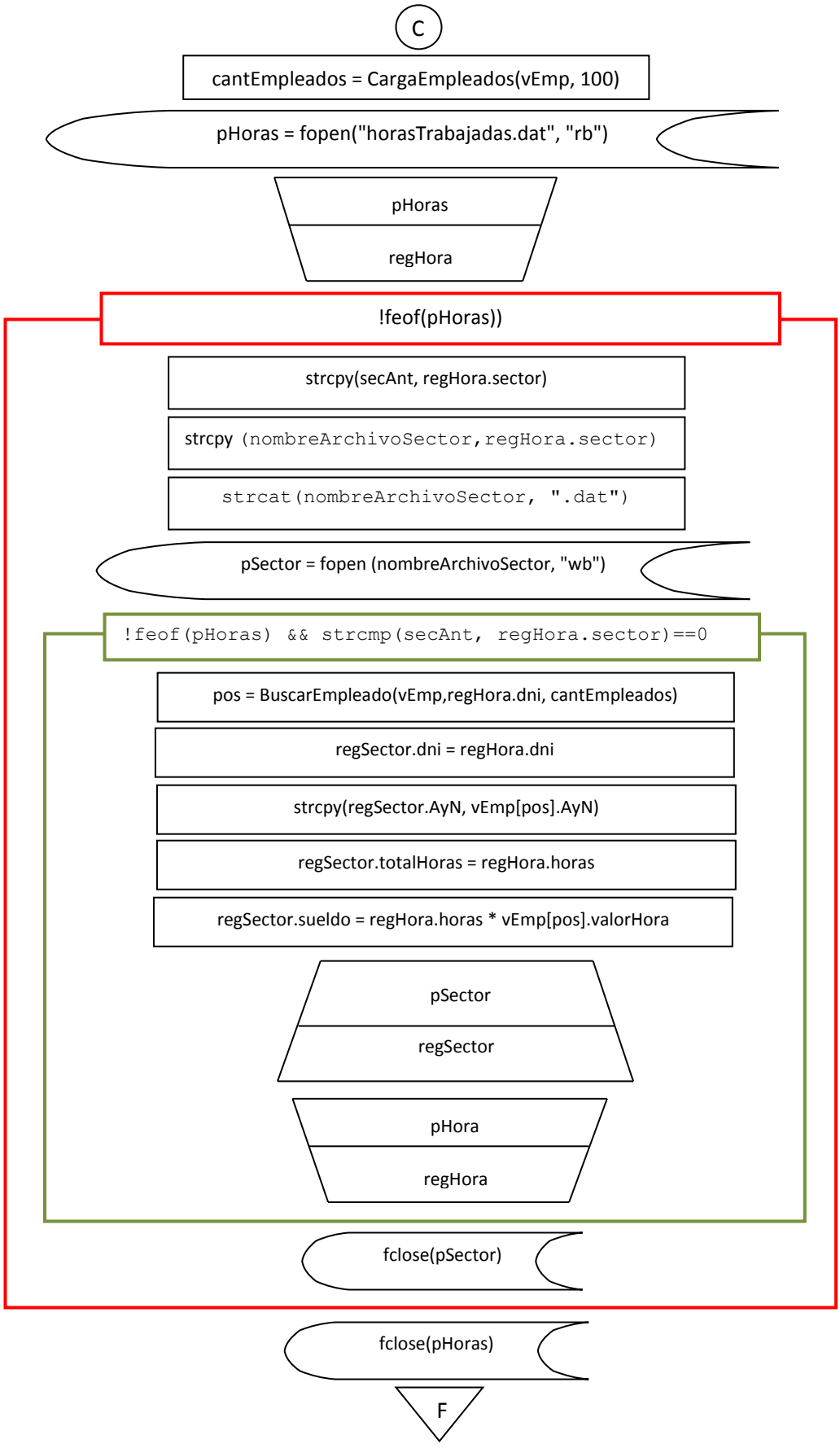
Se necesitan tres estructuras una por cada archivo (en realidad para los sectores se crean varios archivos, pero todos con la misma estructura):

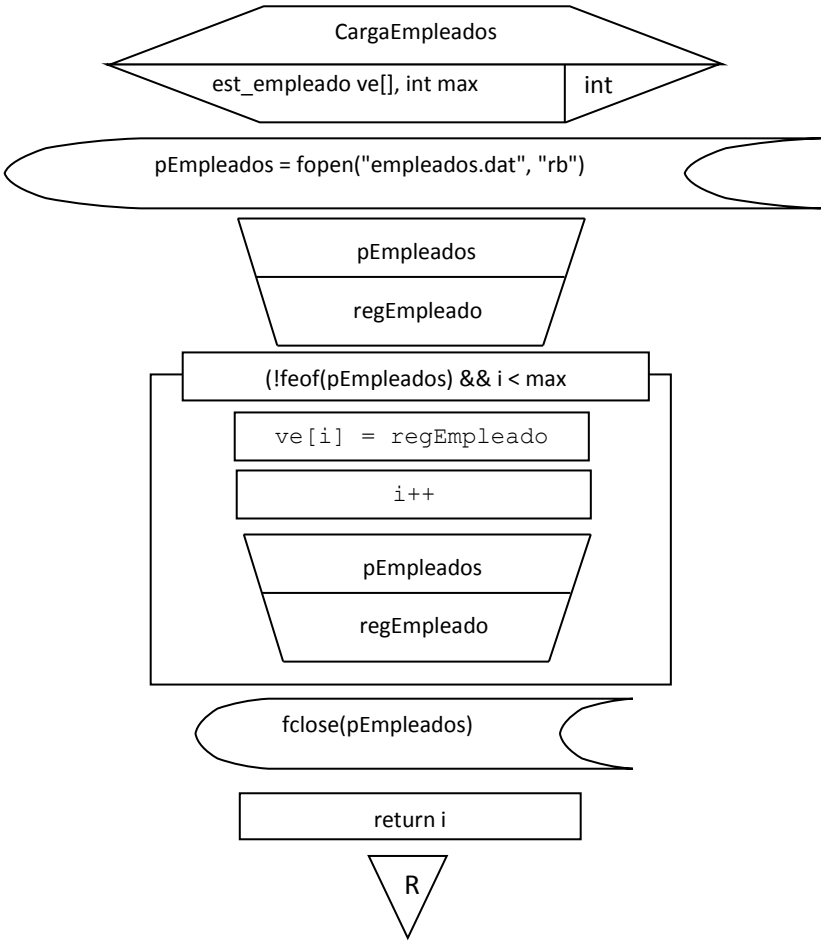
Estructura **est_horas**		
sector	dni	horas

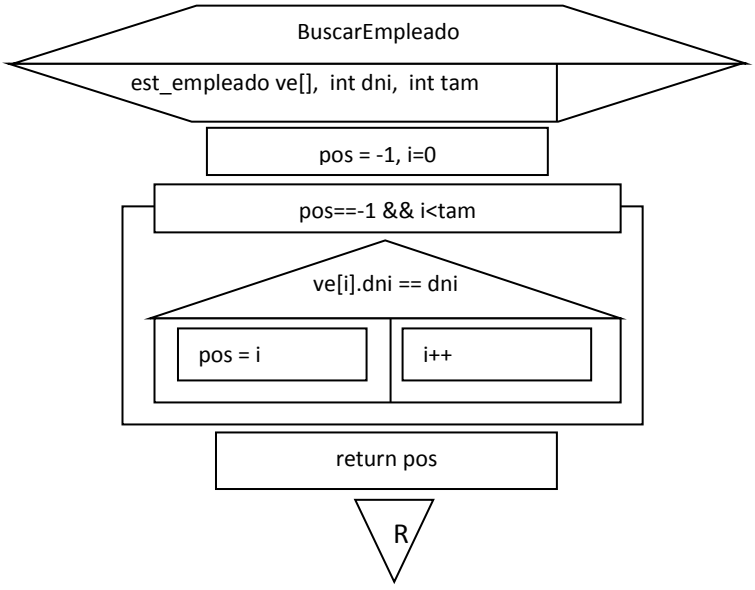
Estructura **est_empleado**		
dni	AyN	valorHora

Estructura **est_sector**			
dni	AyN	totalHoras	sueldo

Diagrama de Lógica:




CargaEmpleados
est_empleado ve[], int max
int
pEmpleados = fopen("empleados.dat", "rb")
pEmpleados
regEmpleado
(!feof(pEmpleados) && i < max
ve[i] = regEmpleado
i++
pEmpleados
regEmpleado
fclose(pEmpleados)
return i
R


BuscarEmpleado
est_empleado ve[], int dni, int tam
pos = -1, i=0
pos==-1 && i<tam
ve[i].dni == dni
pos = i
i++
return pos
R

Codificación en Lenguaje C:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

typedef struct
{
    char sector[11];
    int dni;
    int horas;
} est_horas;

typedef struct
{
    int dni;
    char AyN[31];
    float valorHora;
} est_empleado;

typedef struct
{
    int dni;
    char AyN[31];
    int totalHoras;
    float sueldo;
} est_sector;

int CargaEmpleados (est_empleado[],int);
int BuscarEmpleado (est_empleado[],int,int);

int main()
{
    FILE * pHoras, * pSector;
    int cantEmpleados, pos;
    est_empleado vEmp[100];
    est_horas regHora;
    est_sector regSector;
    char secAnt[11], nombreArchivoSector[15];

    cantEmpleados = CargaEmpleados(vEmp, 100);

    pHoras = fopen("horasTrabajadas.dat", "rb");
    if (pHoras == NULL)
    {
        printf ("No se pudo abrir el archivo de horas trabajadas");
        getch();
        exit (1);
    }

    fread(&regHora, sizeof(regHora), 1, pHoras);
    while (!feof(pHoras))
    {
        strcpy(secAnt, regHora.sector);
        /*Se genera el nombre del archivo basado en el nombre del sector */
        strcpy (nombreArchivoSector, regHora.sector);
        strcat(nombreArchivoSector, ".dat");
        pSector = fopen (nombreArchivoSector, "wb");
        if (pSector ==NULL)
        {
            printf ("No se pudo crear el archivo para el sector %s", secAnt);
            getch();
            exit(1);
        }

        while (!feof(pHoras) && strcmp(secAnt, regHora.sector)==0)
        {
            /*se busca el empleado y como se trabaja con archivos ya creados
            se asume que siempre lo encuentra*/
```

```
            pos = BuscarEmpleado(vEmp,regHora.dni, cantEmpleados);
            regSector.dni = regHora.dni;
            strcpy(regSector.AyN, vEmp[pos].AyN);
            regSector.totalHoras = regHora.horas;
            regSector.sueldo = regHora.horas * vEmp[pos].valorHora;
            fwrite(&regSector, sizeof(regSector),1 , pSector);

            fread(&regHora, sizeof(regHora), 1, pHoras);
        }

        fclose(pSector);
    }
    fclose(pHoras);
}

int CargaEmpleados (est_empleado ve[],int max)
{
    FILE *pEmpleados;
    int i=0;
    est_empleado regEmpleado;
    pEmpleados = fopen("empleados.dat", "rb");
    if (pEmpleados == NULL)
    {
        printf ("No se pudo abrir el archivo de empleados.");
        getch();
        exit (1);
    }

    fread(&regEmpleado, sizeof(regEmpleado),1, pEmpleados);
    while (!feof(pEmpleados) && i < max)
    {
        ve[i] = regEmpleado;
        i++;
        fread(&regEmpleado, sizeof(regEmpleado),1, pEmpleados);
    }
    fclose(pEmpleados);
    return i;
}

int BuscarEmpleado (est_empleado ve[],int dni,int tam)
{
    int i =0 , pos =-1;
    while (pos==-1 && i<tam)
    {
        if (ve[i].dni == dni)
            pos = i;
        else
            i++;
    }
    return pos;
}
```