

Patrones de resolución con vectores

En este documento comenzamos a trabajar con patrones de resolución de algoritmos, utilizando las estructuras de datos arreglos y registros.

Temas a desarrollar

- Corte de control
- Mezcla
- Apareo

Antes de comenzar con la ejercitación hagamos un resumen

Las estructuras o registros permiten agrupar datos de diversos tipos asociados a una entidad en particular. Por ejemplo las personas tienen documento, nombre, apellido, fecha de nacimiento, etc. Es muy útil para una buena comprensión del código y permite el pasaje de parámetros en forma clara y ordenada.

Los arreglos son estructuras de datos homogéneas (todos sus datos son del mismo tipo) que permiten almacenar un determinado número de datos bajo un mismo identificador, para luego referirse a los mismos utilizando uno o más subíndices. Los arreglos pueden pensarse como vectores, matrices, etc. Para poder utilizar un arreglo, primero es obligatorio su dimensionamiento; es decir, definirlo declarando los rangos de sus subíndices, lo cual determina cuántos elementos se almacenarán y cómo se accederá a los mismos. Todas las celdas de un arreglo son del mismo tipo, este puede ser un tipo simple (int, float, string, char, etc), un tipo complejo definido por el desarrollador (otro arreglo, una estructura, etc)

Hay que tener en cuenta que se utilizan arreglos, de 1 o 2 dimensiones, toda vez que se debe mantener en memoria varios datos de una misma característica, y se accede a los mismos a través de un índice. El índice varía entre 0 y la dimensión del arreglo menos 1.

Los arreglos son estructuras estáticas, por lo tanto se debe definir durante el desarrollo del programa.

Recordemos la importancia de desarrollar funciones generales para trabajar con arreglos. Sabemos la dificultad de resolver un algoritmo, y la importancia de descomponer en módulos la solución "divide y triunfarás".

Así las cosas, tener resueltas algunas "herramientas" antes de comenzar a desarrollar cualquier solución, propende a encarar el problema con la "mente" puesta en el tratamiento de los datos y no en el cómo.

Es pensar algo así cómo **¿qué se debe hacer con estos datos para arribar al resultado?**, y NO pensar en **¿cómo lo debemos hacer?**

Qué queremos decir. Si se trata de buscar cuántas veces se repite un valor en una lista, se debe buscar el valor. Si NO se lo encuentra se informa que se repite 0 veces, en otro caso se debe seguir buscando y contando cada vez que aparece.

Solución mediante pseudocódigo.

```
función contar <- ContarV(vector, valor)
  definir contar como entero
  lugar <- buscarV(vector, 0, tamaño, valor)    // busca en el vector desde la posición 0 hasta el final o hasta que encuentra el valor
  contar <- 0
  si (lugar>-1) entonces
    desde <- lugar + 1
    repetir
      contar <- contar + 1
      lugar <- buscarV(vector, lugar+1, tamaño, valor)
    hasta que (lugar== -1)
  finsí
finfuncion
```

Entonces, si modificamos la función `buscarV` del módulo anterior, podemos pensar una función `contarV` que utilice la anterior y luego usar uno, otra o ambas en diversos problemas .

Patrón de resolución Corte de control

¿Qué es Corte de Control?

El corte de control es un proceso que se puede aplicar a cualquier lista de datos, con la única condición que esté ordenada, que permite en forma clara y consensuada procesar en categorías determinadas por los criterios de ordenamiento.

En otras palabras, se procesa un conjunto ordenado de registros en subconjuntos determinados por los criterios de orden.

Siguiendo en el mismo ejemplo de las clases anteriores, se cuenta con una lista de los alumnos de la facultad que contienen –entre otros datos- el código de la carrera y el año de ingreso.

La lista está ordenada por esos dos campos: código de carrera; año de ingreso. Esto quiere decir que están primero aquellos estudiantes de la carrera cuyo código sea el menor, a continuación los estudiantes de la carrera con el código siguiente (en orden) y así sucesivamente. Además en cada grupo de estudiantes (por carrera) están ordenados por año de ingreso (de menor a mayor)

Se requiere generar un reporte separado por carrera y en cada una ordenado por el año de ingreso.

Este es un problema clásico y está muy estudiado. Todos los desarrolladores lo resuelven mediante el mismo patrón. Esto permite una solución uniforme y probada. Está basada en el orden de los datos, por lo cual si los datos no lo están se deben ordenar antes de aplicar esta solución

Fuente: artículo publicado por el Ing. Leonardo Secotaro <http://lsecotaro.blogspot.com/2011/09/corte-de-control.html>

Consideraciones a tener en cuenta para resolver el algoritmo de Corte de Control

Realizar un corte de control significa que se deben **agrupar** y **acumular** por un criterio determinado **llamado clave** los datos a procesar.

Esto significa que mientras la clave sea la misma, se deben procesar los datos (lo podemos denominar **proceso general**) y cuando la **clave** cambia se realiza algún otro proceso y luego se retorna al **proceso general** de los datos.

Un corte de control puede tener uno o más niveles de corte

Para procesar, por ejemplo, una lista de alumnos ordenada por carrera, donde se quieren mostrar todos los alumnos por carrera y el total de cada carrera (al finalizar cada una), se realiza un corte de control de nivel 1. Si los datos están ordenados por carrera y dentro de esta por año de ingreso y se quieren agregar totales por cada año de ingreso, estamos frente a un problema de corte de control de nivel 2. Y así sucesivamente (hacia afuera podemos pensar en regional)

Para lograr **controlar** el cambio se necesita "**recordar**" la clave que se está procesando. A esta clave se la puede identificar mediante la variable "**claveNAnterior**".

Donde N un número que significa que clave es, la primera "1", la segunda "2", la enésima "N"; otra nomenclatura más clara es el nombre de esa clave, por ejemplo si la clave es "carrera", la variable se llamará "carreraAnterior".

Parece difícil, pero es súper fácil.

Veremos a continuación una estructura o esquema general, luego el código en C++ y algunas pruebas del programa.

Debemos trabajar con paciencia y analizar el problema. Una buena estrategia es escribir en una hoja (puede ser de papel o una hoja de una planilla de cálculo) una lista con los datos de las carreras y estudiantes, ordenada por carrera y analizar sobre la misma cómo se hace el tratamiento de los datos.

Estructura o esquema general del algoritmo de corte de control

```
[Traer primer dato]
while(condicion_fin_recorrido) {
    clave1_anterior = clave1Actual
    [inicializar contadores para el primer corte]
    while(condicion_fin_recorrido Y clave1_anterior ==
        clave1Actual) {
        clave2_anterior = clave2Actual
        [inicializar contadores para el segundo corte]
        while(condicion_fin_recorrido Y
            clave1_anterior == clave1Actual
            AND clave2_anterior == clave2Actual) {
            [incrementar contadores del segundo corte]
            [escribir detalle del segundo corte]
            [traer datos siguiente]
        }
        [escribir totales acumulados del segundo corte]
        [incrementar contadores del primer corte]
    }
    [escribir totales acumulados del primer corte]
    [incrementar contadores generales]
}
[escribir totales generales]
```

DONDE

1. Comenzamos con una “lectura adelantada”, u obtener el primer dato.
2. Mediante un ciclo mientras que controla el fin de los datos se controla todo el proceso. Si hay más de un nivel, para cada uno se incluye un ciclo mientras que arrastra la condición la condición de finalización e incluye la nueva condición de corte.
3. Dentro del ciclo más interno de debe avanzar con los datos y se denomina lectura siguiente.
4. Por cada corte se definen acumuladores y contadores que se modifican de acuerdo al problema a resolver. Se deben inicializar en el nivel anterior.
5. Las claves anteriores se deben asignar en el nivel anterior.
6. Al salir de cada ciclo se pueden imprimir los totales y acumular para los niveles anteriores.

Prestemos mucha atención al esquema general. Es un proceso clásico que se va repitiendo por cada nivel de corte que se agrega

Veamos el código en C++ con un problema

Procesar los datos de estudiantes de la facultad, que contienen los siguientes datos: DNI., nombre, apellido, carrera, año de ingreso. Se deben mostrar los datos ordenados por carrera / año de ingreso. Al final se deben mostrar totales de ingresantes por cada carrera y total de estudiantes de la facultad.

Tener en cuenta que se debe procesar la información ordenada por los campos de "corte". En este ejemplo la información se ordena por carrera. Usamos un método de ordenamiento Shell adaptado a ordenar por dos campos: nombre de la carrera, año de inscripción.

Ordenamos por carrera / año para un segundo ejercicio con 2 cortes.

```
void cambio( tRegEst & A, tRegEst & B){
    tRegEst aux;
    aux = A;
    A = B;
    B = aux;
}
```

```
void ordenarVEst(tVectorEst vector, int cantVector){
    int i , j , k , salto ;   bool fin ;

    salto = cantVector ;
    while ( salto > 0 ) {
        salto = salto / 2 ;
        do {
            fin = true ;
            k = cantVector - salto ;
            for ( i = 0 ; i < k ; i++ ) {
                j = i + salto ;
                if ( strcmp(vector[i].carrera, vector[j].carrera)>0) {
                    cambio(vector[i], vector[j]);
                    fin = false ;
                } else
                    if (( strcmp(vector[i].carrera, vector[j].carrera)==0)
                        && (vector[i].anioIngreso>vector[j].anioIngreso)) {
                        cambio(vector[i], vector [j]);
                        fin = false ;
                    }
            }
        } while ( ! fin ) ;
    }
}
```

Se compara la carrera y en caso que estén desordenadas, se intercambian

Se compara la carrera y en caso sean iguales se compara el año y en caso que estén desordenadas, se intercambian


```

void cargarV(tVectorEst v, int & cant) {
    int i=0 ;
    int dni;
    printf("Ingresar DNI(0=fin) ");
    cin>>dni;
    while ((i<cant) && (dni!=0)) {
        v[i].DNI = dni;
        printf("Ingresar nombre ");
        fflush(stdin);
        cin.getline(v[i].nombre, sizeof(v[i].nombre),'\n');
        printf("Ingresar apellido ");
        fflush(stdin);
        cin.getline(v[i].apellido, sizeof(v[i].apellido),'\n');
        printf("Ingresar carrera ");
        fflush(stdin);
        cin.getline(v[i].carrera, sizeof(v[i].carrera),'\n');
        printf("Ingresar año ingreso ");
        cin>>v[i].anioIngreso;
        i++ ;
        printf("Ingresar DNI(0=fin) ");
        cin>>dni;
    }
    cant = i ;
};

```

Para el pedido de datos se utiliza la función getline que permite ingresar frases. La función fflush "limpia" el buffer del teclado.

Mediante la función printf se puede "formatear" la salida.

```

void mostrarV(tVectorEst v, int cant) {
    printf("%8s %-25s %-25s %-25s %4s\n",
        "DNI", "Nombre", "Apellido", "Carrera", "Año");
    for (int i=0;i<cant;i++)
        printf("%8i %-25s %-25s %-25s %-i\n",
            v[i].DNI, v[i].nombre, v[i].apellido,
            v[i].carrera, v[i].anioIngreso );
    system("pause") ;
};

```

Mediante la strcpy se asignan valores a variables de tipo caracter.

```

void inicializarV(tVectorEst v, int cant) {
    for (int i=0;i<cant;i++) {
        v[i].DNI = 0 ;
        strcpy(v[i].nombre, "");
        strcpy(v[i].carrera, "");
        strcpy(v[i].apellido, "");
        v[i].anioIngreso= 0 ;
    }
};

```

```

#include <iostream>
#include <string.h>
using namespace std;
#define _CANT 50
typedef char str30[30] ;
typedef struct tRegEst {
    long int DNI;
    str30 carrera;
    int anioIngreso;
    str30 apellido ;
    str30 nombre ;
};
typedef tRegEst tVectorEst[_CANT]
;

int main(){
    tVectorEst vEst ;
    int cantEst = _CANT;
    setlocale(LC_CTYPE, "Spanish");
    inicializarV(vEst,_CANT) ;
    cargarV(vEst,cantEst) ;
    ordenarVEst(vEst,cantEst);
    mostrarV(vEst,cantEst) ;
    procesarCteCtrolV (vEst,cantEst) ;
}

```

Para que el programa utilice palabras con eñe y tilde

Se ordenan los datos por carrera / año ingreso

Este proceso realiza el corte de control por carrera / año

Datos a procesar

12	Juan	Valdez	Ing Sistemas	2015
13	Analía	Andrada	Ing Alimentos	2018
14	Carlos	Martinez	Ing Sistemas	2014
15	María	Gutierrez	Ing Sistemas	2016
16	Marco	Antonio	Ing Alimentos	2012
17	Mara	Gimenez	Ing Industrial	2018
18	Julieta	Martina	Ing Industrial	2013
19	Facundo	Arana	Ing Sistemas	2017
20	Marina	Mares	Ing Alimentos	2016
21	Mariano	Martinez	Ing Sistemas	2012
22	Diego	Armando	Ing Industrial	2006
0				

Listado de estudiantes procesado

16	Marco	Antonio	Ing Alimentos	2012
20	Marina	Mares	Ing Alimentos	2016
13	Analía	Andrada	Ing Alimentos	2018
22	Diego	Armando	Ing Industrial	2006
18	Julieta	Martina	Ing Industrial	2013
17	Mara	Gimenez	Ing Industrial	2018
21	Mariano	Martinez	Ing Sistemas	2012
14	Carlos	Martinez	Ing Sistemas	2014
12	Juan	Valdez	Ing Sistemas	2015
15	María	Gutierrez	Ing Sistemas	2016
19	Facundo	Arana	Ing Sistemas	2017

Corte de control de un nivel: carrera

```
void procesarCteCtrlV (tVectorEst vEst, int cant) {
    str30 carAnt;
    int i = 0;
    tRegEst r;
    bool fin;
    int total = 0 ;
    int totalCar ;
    traer(vEst, r, cant, i, fin) ;
    while (!fin) {
        strncpy(carAnt, r.carrera, sizeof(carAnt)) ;
        totalCar = 0 ;
        printf("Carrera %s\n", carAnt);
        while ((!fin ) && (strcmp(carAnt, r.carrera)!=0)) {
            totalCar ++;
            traer(vEst, r, cant, i, fin) ;
        }
        printf("Total carrera  %d\n",  totalCar);
        total = total + totalCar ;
    }
    printf("total ingreso de las carreras es  %d\n", total);
}
```

Mediante esta función se accede a los datos de forma secuencial. Es importante trabajar con esta metodología dado que generaliza la solución.

```
void traer(tVectorEst v, tRegEst & r, int cant, int & ptro, bool & fin) {
    if (ptro < cant) {
        r = v[ptro];
        ptro++;
        fin = false ;
    } else fin = true;
}
```

En el algoritmo hay un nivel de corte, que se resuelve mediante dos ciclos:

- el más externo llamado general que controla el fin de datos
- el más interior controla el fin de datos y todos los campos de corte (carrera y el fin de datos)

Hay que tener en cuenta que las condiciones de fin de ciclo se deben replicar hacia adentro del algoritmo. Se debe a que el cambio de datos sucede en la parte más central del proceso.

Corte de control de dos niveles: carrera, año

```
void procesarCteCtrolV (tVectorEst vEst, int cant) {
    str30 carAnt;
    int anioAnt;
    int i = 0;
    tRegEst r;
    bool fin;
    int total = 0 ;
    int totalCar ;
    int totalAnio ;
    traer(vEst, r, cant, i, fin) ;
    while (!fin) {
        strncpy(carAnt, r.carrera, sizeof(carAnt)) ;
        totalCar = 0 ;
        printf("Carrera %s\n", carAnt);
        while ((!fin ) && (strcmp(carAnt, r.carrera)==0)) {
            anioAnt = r.aniIngreso;
            totalAnio = 0 ;
            while ((!fin ) && (strcmp(carAnt, r.carrera)==0) &&
                (anioAnt == r.aniIngreso)) {
                totalAnio ++;

                traer(vEst, r, cant, i, fin) ;
            }
            printf("    Año %d  %d\n", anioAnt, totalAnio);
            totalCar = totalCar + totalAnio;
        }
        printf("Total carrera  %d\n",  totalCar);
        total = total + totalCar ;
    }
    printf("total ingreso de las carreras es  %d\n", total);
}
```

Mediante esta función se accede a los datos de forma secuencial. Es importante trabajar con esta metodología dado que generaliza la solución.

```
void traer(tVectorEst v, tRegEst & r, int cant, int & ptro, bool & fin) {
    if (ptro < cant) {
        r = v[ptro];
        ptro++;
        fin = false ;
    } else fin = true;
}
```

En el algoritmo hay dos niveles de corte, que se resuelve mediante tres ciclos:

- el más externo llamado general que controla el fin de datos
- el más interior controla el fin de datos y todos los campos de corte (carrera, año ingreso)
- el del "medio" que controla el cambio de carrera y el fin de datos

Hay que tener en cuenta que las condiciones de fin de ciclo se deben replicar hacia adentro del algoritmo. Se debe a que el cambio de datos sucede en la parte más central del proceso.

Resultados de los corte de control

Resultado
corte de
control de un
nivel

Ing Alimentos	
Total carrera	3
Ing Industrial	
Total carrera	3
Ing Sistemas	
Total carrera	10

Resultado
corte de
control ce dos
niveles

Ing Alimentos			
Año	2012	1	
Año	2016	1	
Año	2018	1	
Total carrera	3		
Ing Industrial			
Año	2006	1	
Año	2013	1	
Año	2018	1	
Total carrera	3		
Ing Sistemas			
Año	2014	1	
Año	2015	1	
Año	2016	1	
Año	2017	1	
Total carrera	10		

Vemos en los resultados que a medida que se agregan niveles de corte, se logra un mayor detalle en la planilla.

Resolver problemas mediante el patrón corte de control es una tarea relativamente fácil, permite focalizar resultados en forma ordenada y desagregada.

Es importante utilizar este patrón, dado que está estandarizado y lo comprender todos.

Patrón de resolución Apareo

¿Qué es apareo?

El apareo es un proceso que se puede aplicar a cualquier par de listas de datos, con la única condición que estén ordenadas, que permite en forma clara y consensuada unir las dos listas en una sola.

En otras palabras, se procesa un conjunto ordenado de registros de varias listas, logrando tener una sola sin tener que volver a ordenarla.

Siguiendo en el mismo ejemplo de las clases anteriores, se cuenta con una lista de los alumnos de la facultad que contienen –entre otros datos- el código de la carrera y el año de ingreso. Y una segunda lista con los ingresantes en el último año.

La lista está ordenada por esos dos campos: código de carrera; año de ingreso. Esto quiere decir que están primero aquellos estudiantes de la carrera cuyo código sea el menor, a continuación los estudiantes de la carrera con el código siguiente (en orden) y así sucesivamente. Además en cada grupo de estudiantes (por carrera) están ordenados por año de ingreso (de menor a mayor)

Se requiere generar una única lista **NUEVA** que contenga las dos anteriores y conserve el orden.

Este es un problema clásico y está muy estudiado. Todos los desarrolladores lo resuelven mediante el mismo patrón. Esto permite una solución uniforme y probada. Está basada en el orden de los datos, por lo cual si los datos no lo están se deben ordenar antes de aplicar esta solución

Consideraciones a tener en cuenta para resolver el algoritmo de Apareo

Aparear dos listas es un concepto que se puede aplicar en diversos casos:

1. Se tienen dos listas, por ejemplo de números, con datos que no se repiten, ordenadas. Se quiere generar una nueva lista que contenga todos los números y que mantenga el orden (no se la deba volver a ordenar). Esto se denomina **MEZCLA**
2. El proceso anterior se puede generalizar a N listas, en un único algoritmo (resolver todas las listas juntas, NO de a dos)
3. Se tienen dos listas, la primera se la denomina **MAESTRO** y la segunda **NOVEDADES**. Todos los datos de la segunda (por la clave) están en la primera, ambas listas están ordenadas por la misma clave. Por ejemplo se tiene la lista de artículos de un supermercado y la lista de venta de los artículos. Se debe sumar en la lista de artículos la cantidad de venta de cada uno de estos. Esto se denomina **ACTUALIZACIÓN SIN ALTAS NI BAJAS**. Este proceso se lo puede generalizar a N listas de NOVEDADES
4. Se tienen dos listas, la primera se la denomina **MAESTRO** y la segunda **NOVEDADES**, ambas listas están ordenadas por la misma clave. Por ejemplo se tiene la lista de estudiantes y la lista de las novedades del año 2020, con las nuevas altas, la información de aquellos que se dan de baja y los cambios en los datos de los estudiantes (número de teléfono, dirección, mail, etc). Se debe ACTUALIZAR el maestro con las novedades. Esto se denomina **ACTUALIZACIÓN**. Este proceso se lo puede generalizar a N listas de NOVEDADES

Parece difícil, pero es súper fácil.

EN TODOS LOS CASOS SE TRATA DE GENERAR UNA NUEVA LISTA

Veremos a continuación una estructura o esquema general, luego el código en C++ y algunas pruebas del programa.

Estructura o esquema general del algoritmo de apareo Mezcla

```
[Traer primer dato maestro]
[Traer primer dato novedad]

while(condicion_fin_recorrido Maestro y condicion_fin_novedad ) {
    si (clave maestro < clave novedad)
        [pasar a la lista de salida los datos del maestro]
        [traer siguiente datos maestro]
    sino
        si (clave maestro == clave novedad)
            [pasar a la lista de salida los datos ]
            [traer siguiente datos maestro]
            [traer siguiente datos novedad]
        sino
            [pasar a la lista de salida los datos de la novedad]
            [traer siguiente datos novedad]
        finsi
    finsi
}
si (NO (condicion_fin_recorrido Maestro))
    [procesar el resto de los registros del maestro]
sino
    [procesar el resto de los registros de las novedades]
}
```

DONDE

1. Comenzamos con una “lectura adelantada”, de ambas listas.
2. Mediante un ciclo mientras que controla el fin de los datos se controla todo el proceso. Prestar atención que se controlan ambas listas
3. Se analizan las claves, si la clave del registro del maestro es menor a la clave del registro de novedad, se pasa el maestro a la lista de salida. Si es menor la clave de la novedad, se pasa esta a la lista de salida. Mientras que si son iguales se procesa y se pasa (cualquiera de las dos) a la lista de salida.
4. De acuerdo a que registro se pase (MAESTRO o NOVEDAD) se "avanza" en esa lista
5. Cuando finaliza alguna de las listas se debe procesar el remanente de la otra. Puede ser MAESTRO o NOVEDAD

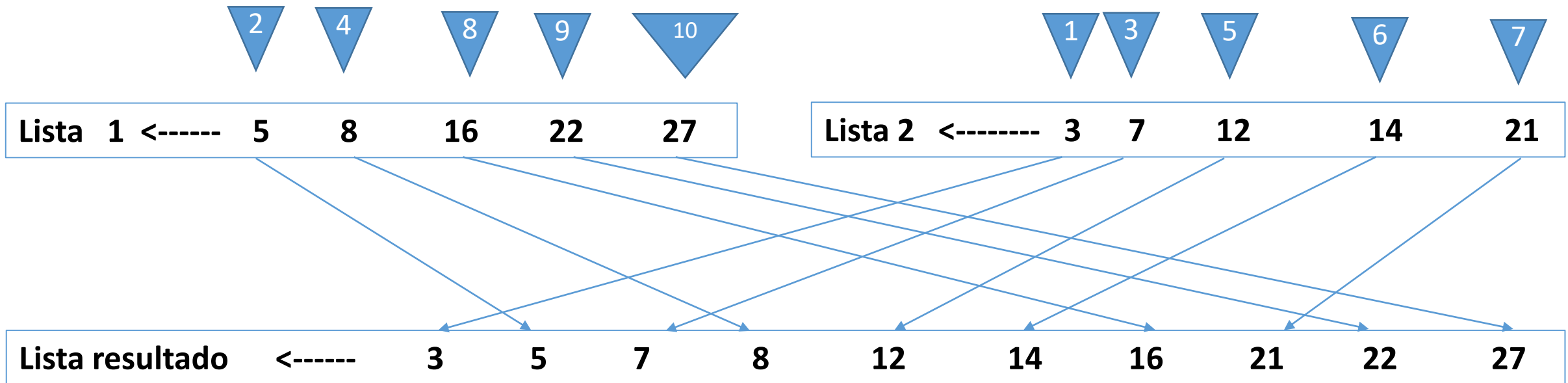
Prestemos mucha atención al esquema general. Es un proceso clásico que se va repitiendo por cada nivel de corte que se agrega

Veamos el código en C++ con un problema

Procesar dos listas de números y generar una nueva lista que contenga todos los números de la primer lista y todos los números de la segunda lista, manteniendo el orden. Ambas listas deben estar ordenadas, caso contrario se ordenan en este algoritmo.

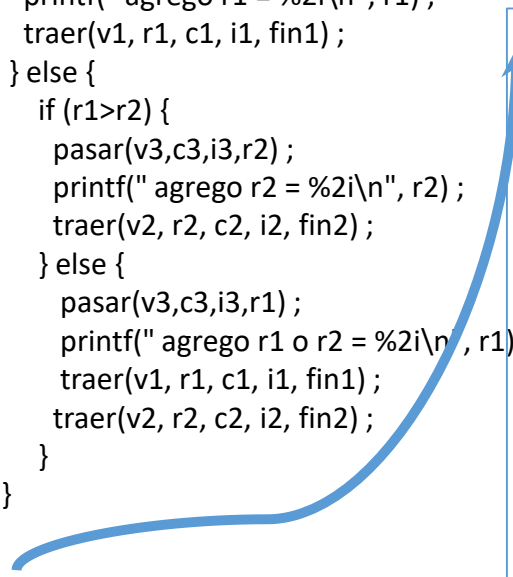
Los procesos de ordenamiento, principal, pedido de datos, etc., son similares al ejercicio de corte de control. Por tal motivo no lo agregamos en este problema.

A continuación vemos en un ejemplo el contenido de cada lista, el orden en que se procesan los datos y cómo debe quedar la lista resultado.



Apareo, Mezcla

```
void apareo (tVector v1, int c1, tVector v2, int c2, tVector v3, int & c3) {
    int i1=0;
    int i2=0;
    int i3=0;
    int r1, r2, r3;
    bool fin1, fin2;
    traer(v1, r1, c1, i1, fin1);
    traer(v2, r2, c2, i2, fin2);
    while((!fin1)&&(!fin2)){
        printf("r1 = %2i  r2=%2i", r1, r2);
        if (r1<r2) {
            pasar(v3,c3,i3,r1);
            printf(" agrego r1 = %2i\n", r1);
            traer(v1, r1, c1, i1, fin1);
        } else {
            if (r1>r2) {
                pasar(v3,c3,i3,r2);
                printf(" agrego r2 = %2i\n", r2);
                traer(v2, r2, c2, i2, fin2);
            } else {
                pasar(v3,c3,i3,r1);
                printf(" agrego r1 o r2 = %2i\n", r1);
                traer(v1, r1, c1, i1, fin1);
                traer(v2, r2, c2, i2, fin2);
            }
        }
    }
}
```



```
.....
if (!fin1) {
    do {
        pasar(v3,c3,i3,r1);
        traer(v1, r1, c1, i1, fin1);
    } while (!fin1);
} else {
    do {
        pasar(v3,c3,i3,r2);
        traer(v2, r2, c2, i2, fin2);
    } while (!fin2);
}
c3 = i3;
}
```

El algoritmo const de dos partes:

- procesar ambas listas mientras haya elementos en las dos.
- Cuando finaliza una de las listas se pasa los datos de la restante a la lista resultado

Mediante las funciones **traer** y **pasar** se "manipula" las listas. Esta forma de trabajar permite "distanciarse" del origen y destino de los datos.

¿Qué quiere decir "manipula" y "distanciarse"?

Se trata de traer o poner los datos en "algún lugar", sin importar dónde. Para hacer el proceso de corte de control o de apareo no debería importar el origen y destino de los datos. Se le da el control a funciones para que obtengan i guarden datos.

Más adelante trabajaremos con archivos en disco y estos procesos no van a requerir cambios (salvo los procesos traer y pasar).

Para finalizar pensemos en los otros tipos de apareo mencionados a través de ejemplos:

1. Lista maestro con datos del estudiante (legajo, datos personales, cantidad de materias aprobadas); lista de novedades con datos de materias aprobadas de los estudiantes (legajo, cantidad de materias). Se trata de "actualizar" la lista de alumnos (maestro) con la cantidad de materias aprobadas (novedades).
Es un caso particular de la mezcla, donde no hay novedades sin maestro (el caso $r1 > r2$ no puede suceder). Hay novedades para alguno/s de lo/s estudiante/s.
2. Lista maestros con los datos de los estudiantes (legajo, datos personales); lista de novedades con modificaciones a los datos del maestro (domicilio, teléfono, mail). En este caso las novedades deben tener un campo que indique el tipo de novedad (Alta, Baja, Cambio), que pueden ser estudiantes de baja, nuevos estudiantes o estudiantes a quienes se les modifica algún dato (no puede ser el legajo). En este caso se debe analizar en el algoritmo presentado qué hacer en los casos menor, mayor o igual y ver el tipo de novedad
3. Otra posibilidad es permitir que la clave de las novedades se repitan. Este es un problema un poco más difícil de resolver, pues no se puede pasar a la lista de salida por cada novedad, sino se debe hacer solamente en caso que la clave del maestro sea mayor a la clave de la novedad. Para lograr esto en una forma ordenada y estandarizada, se trabaja sobre la función *traer*, la cual es diferente para el maestro (cambia) y para la novedad (es la misma). Es un algoritmo un poco más elaborado que el anterior.
4. Aparear una lista maestro con varias listas de novedades. Este problema es similar al resuelto en este documento. Se deben resolver dos funciones traer diferentes: para el maestro (es la misma) y para las novedades (cambia). La función traer de novedades debe "mirar" el primero de cada lista de novedades y traer el más chico. Esto se resuelve con un vector de claves de novedades, con una celda por cada lista que contiene el próximo a traer de cada lista. Entonces la función *traer* Novedad deben tener traídos (al inicio del algoritmo) un registro de cada lista de novedades, buscar la menor clave y devolver ese registro. Tener presente que el registro que se devuelve (por ejemplo de la lista 2) debe ser actualizado en el vector (en el ejemplo traer del 2).