TP N°2: Operaciones entre archivos.

Alumno: Juan Cruz Mateos
Mat. 15134

Enunciado 1

Se quiere **registrar todas las notas de un alumno de una facultad.** Para esto se arma un registro con Código de la Universidad, Código de Facultad, Código de la Asignatura, Legajo del Alumno, Edad al momento de rendir, Código del Docente a cargo, Nota y Observaciones.

Enunciado 2

Se quiere **registrar todas las compras de un local nacional de electrodomésticos**. Para esto se realiza un registro de la siguiente manera: Fecha, Hora, Código de Provincia, Código de Ciudad, Código del Local, Legajo del Vendedor, Factura, Código del Producto, Cantidad, Valor Unitario, Total Renglón y Observaciones.

Enunciado 3

Se tiene un archivo maestro y de novedades (ambos ordenados por ID) con la siguiente estructura: **ID, Descripción, valor**.

1) Para los enunciados 1 y 2 realizar las estructuras lógicas, tanto en formato fijo como variable.

Enunciado #1: *Identificador :: cod_asig, legajo, nota. (pueden existir cod_asig y legajos repetidos ya que el alumno puede desaprobar un final y aprobarlo en una instancia posterior, pero los registros se podran diferenciar por el campo nota).*

Longitud Fija:

```
Examen (
     cod_univ: E4,
     cod_fac: E4,
     cod_asig: E4,
     legajo: E4,
     edad: E1,
     cod_doc: E4,
     nota: E1,
     obs: C100
     )
```

Longitud Variable:

```
Examen (
     cod_univ: E4,
     cod_fac: E4,
     cod_asig: E4,
     legajo: E4,
     edad: E1,
     cod_doc: E4,
     nota: E1,
     obs: T
     )
```

Enunciado #2: *Identificador :: factura.*

Longitud Fija:

```
Compra (
      fecha: C10,
      hora: C5,
      cod_prob: E4,
      cod_ciudad: E4,
      cod_local: E4,
      leg_vend: E4,
      factura: E4,
      cod_prod:E4
      cant: E2,
      valor_un: F8,
      total_rg: F8,
      obs: C100
      )
```

Longitud Variable:

```
Compra (
      fecha: C10,
      hora: C5,
      cod_prob: E4,
      cod_ciudad: E4,
      cod_local: E4,
      leg_vend: E4,
      factura: E4,
      cod_prod:E4
      cant: E2,
      valor_un: F8,
      total_rg: F8,
      obs: T
      )
```

2) Para las estructuras fijas y variables del enunciado 1 realice las consultas de altas, bajas, modificaciones y consultas. Las consultas solicitadas son: promedio con y sin aplazo del alumno, cantidad de materias rendidas y promedio de notas de una materia dada.

Dado que no se indica si el archivo se encuentra ordenado o no, lo supongo desordenado. Por lo tanto:

Estructura fija:

- *altas: append en fin de archivo*
- *bajas: marcado (cambio signo de identificador)*
- *modificacion: recuperar registro y acutalizar*
- *consultas: busqueda lineal*

Pseudocodigos:

***Altas**:*

```
nuevo_registro := resistro
seek(archivo, FIN_ARCHIVO)
write(nuevo_registro, archivo)
```

***Bajas**:*

```
seek(archivo, INICIO_ARCHIVO)
mientras que no fin de archivo && !encontrado:
    read(archivo, reg_act)
    si (cod_asig, legajo y nota coinciden):
        pos = posicion del registro
        encontrado = true
seek(archivo, pos)
reg_act.nota = reg_act.nota * -1
write(reg_act, archivo)
```

```
Obs :: Mantenimiento de bajas: eliminar registros marcados.

open archivo maestro (lectura):  archM
open archivo temp (escritura):   temp

mientras no fin de archM:
        read (registro, archM)
        si registro.nota > 0:
                write(registro, temp)
close archM
close temp
remove archM
rename (temp, archM)
```

Modificaciones:

```
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo && !encontrado:
        read(registro, archivo)
        si (cod_asig, legajo y nota buscados):
                pos = posicion del registro
                encontrado = true
si encontrado:
        seek(archivo, pos)
        write(nuevoregistro, archivo)
```

Promedio con aplazos:

```
total = 0; cant = 0
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo:
        read(registro, archivo)
        si resistro.legajo == legajoAlumno:
                total += registro.nota
                cant += 1

 return cant == 0 ? 0 : total / cant;
```

Promedio sin aplazos:

```
total = 0; cant = 0
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo:
        read(registro, archivo)
        si resistro.legajo == legajoAlumno && registro.nota >= 4:
                total += registro.nota
                cant += 1

 return cant == 0 ? 0 : total / cant
```

Materias rendidas:

```
rendidas := array[50]
n = 0
cant = 0
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo:
        read(registro, archivo)
        si registro.legajo == legajoAlumno:
                i = 0
                mientras i < n && registro.cod_asig != rendidas[i]:
                        i += 1
                si i == n:
                        cant += 1
                        rendidas[n++] = registro.cod_asig;
return cant
```

Promedio Materia

```
total = 0
cant = 0
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo:
        read(registro, archivo)
        si registro.cod_asig == cod_asig:
                total += registro.nota
                cant += 1

return cant == 0 ?  0 ? total / cant
```

3) Para las estructuras fijas y variables del enunciado 2 realice las consultas de altas, bajas, modificaciones y consultas. Las consultas solicitadas son: Promedio de ventas de un vendedor dado, promedio de ventas para una fecha dada, mayor venta de un local, mayor venta de la cadena y promedio de ventas de un producto dado.

Dado que no se indica si el archivo se encuentra ordenado o no, lo supongo desordenado. Por lo tanto:

Estructura fija:

- *altas: append en fin de archivo*
- *bajas: marcado (cambio signo de identificador)*
- *modificacion: recuperar registro y acutalizar*
- *consultas: busqueda lineal*

Pseudocodigos:

***Altas**:*

```
nuevo_registro := resistro
seek(archivo, FIN_ARCHIVO)
write(nuevo_registro, archivo)
```

***Bajas**:*

```
seek(archivo, INICIO_ARCHIVO)
mientras que no fin de archivo && !encontrado:
        read(archivo, reg_act)
        si reg_act.factura == factura:
                pos = posicion del registro
                encontrado = true
seek(archivo, pos)
reg_act. factura = reg_act. factura * -1
write(reg_act, archivo)
```

Obs :: Mantenimiento de bajas: eliminar registros marcados.

```
open archivo maestro (lectura):  archM
open archivo temp (escritura):   temp

mientras no fin de archM:
        read (registro, archM)
        si registro. factura > 0:
                write(registro, temp)
close archM
close temp
remove archM
rename (temp, archM)
```

Modificaciones:

```
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo && !encontrado:
        read(registro, archivo)
        si reg_act.factura == buscado.factura:
                pos = posicion del registro
                encontrado = true
si encontrado:
        seek(archivo, pos)
        write(nuevoregistro, archivo)
```

Promedio ventas vendedor:

```
ventas_vendedor = 0; ventas_totales = 0
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo:
        read(registro, archivo)
        si resistro.leg_vend == legajoVendedor:
                ventas_vendedor += 1
        ventas_totales += 1

 return ventas_total == 0 ? 0 : ventas_vendedor / ventas_totales
```

Promedio ventas fecha:

```
ventas_fecha = 0; ventas_totales = 0
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo:
        read(registro, archivo)
        si resistro.fecha == fecha:
                ventas_fecha += 1
        ventas_totales += 1

 return ventas_totales == 0 ? 0 : ventas_fecha / ventas_totales
```

Mayor venta de un local:

```
mayor_venta := registro
mayor_venta.total_rg = 0;
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo:
        read(registro, archivo)
        si registro.cod_local == cod_local && registro.total_rg > mayor_venta.total_rg:
                mayor_venta = registro
return mayor_venta
```

Mayor venta de una cadena:

```
mayor_venta := registro
mayor_venta.total_rg = 0;
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo:
        read(registro, archivo)
        si registro.total_rg > mayor_venta.total_rg:
                mayor_venta = registro
return registro
```

Promedio ventas de un producto dado:

```
ventas_producto = 0; ventas_totales = 0
seek(archivo, INICIO_ARCHIVO)
mientras no fin de archivo:
        read(archivo, registro)
        si registro.cod_prod == codigoProducto:
                ventas_producto += 1
        ventas_totales +=1
return ventas_totales == 0 ? 0 : ventas_producto / ventas_totales
```

3bis) Ordenar el archivo por algún método, cuáles son las ventajas del método elegido.

Método :: ***Natural Merge Sort***
Caracteristicas ::

- Método de ordenamiento externo util para trabajar con grandes cantidades de informacion (la memoria principal no es suficiente)

- Se trabaja directamente sobre archivos (diferente de Merge Sort, donde se lee parte de la infromacion, se la ordena por algun metodo en memoria principl, se la baja a disco, se repite el proceso con todas las porciones de memoria y luego se mezclan los pedazos ordenados).

- Se utilizan tres archivos: aux1, aux2 y temp.

- El metodo aprovecha la existencia de secuencias ya ordenadas dentro de los datos de los archivos. Si no hay ordenamiento parcial previo el metodo es igualemte efectivo, solo que mas costoso.

4) Apareo: Del enunciado 3 realizar la **intersección**.

Enunciado 3

```
Item (
        id: E4,
        desc: C100,
        valor: F8
      )
```

```
// buscar registros comunes

open archivo maestro open archivo novedades
open archivo interseccion

read(reg_m, maestro)
read(reg_n, novedades)
mientras no fin de maestro && no fin novedades:
    si reg_m.id == reg_n.id:
        si (reg_m.desc == reg_n.desc && reg_m.valor == reg_n.valor):
            write(reg_m, novedades)
        read(reg_m, maestro)
        read(reg_n, novedades)
    else si reg_m.id < reg_n.id:
        read(reg_m, maestro)
    si no:
        read(reg_n, novedades)

close maestro
close novedades
close interseccion
```

5) Apareo: Del enunciado 3 realizar la **diferencia**.

```
// registros solo presentes en maestro (bajas)

open archivo maestro
open archivo novedades
open archivo diferecia

read(reg_m, maestro)
read(reg_n, novedades)
mientras no fin de maestro && no fin novedades:
    si reg_m.id == reg_n.id:
        read(reg_m, maestro)
        read(reg_n, novedades)
    else si reg_m.id < reg_n.id:
```

```
                write(reg_m, diferecia)
                read(reg_m, maestro)
        si no:
                read(reg_n, novedades)

mientras que no fin de maestro:
        read(reg_m, maesrtro)
        write(reg_m, diferecia)

close maestro
close novedades
close diferecia
```

6) Apareo: Del enunciado 3 realizar la **unión**.

```
open archivo maestro
open archivo novedades
open archivo union

read(reg_m, maestro)
read(reg_n, novedades)
mientras no fin de maestro && no fin novedades:
        si reg_m.id == reg_n.id:
                write(reg_m, union)
                read(reg_m, maestro)
                read(reg_n, novedades)
        else si reg_m.id < reg_n.id:
                write(reg_m, union)
                read(reg_m, maestro)
        si no:
                write(reg_n, union)
                read(reg_n, novedades)

mientras que no fin de maestro:
        read(reg_m, maesrtro)
        write(reg_m, union)

mientras que no fin de novedades:
        read(reg_n, novedades)
        write(reg_n, union)

close maestro
close novedades
close union
```

7) Del enunciado 3, realice los cambios necesarios en la estructura del archivo Novedades para que acepte Altas, Bajas y Actualizaciones.

Incorporar un campo codigo que tome los valores 'A', 'B', 'M' según corresponda un alta, baja o modificacion, respectivamente.

```
Item_Novedades (
        id: E4,
        desc: C100,
        valor: F8,
        cod: C1
      )
```

8) Realizar las funciones necesarias para realizar las Altas, Bajas y Actualizaciones del ejercicio 7.

```
open archivo maestro
open archivo novedades
open archivo maestro_act
open archivo errores

read(rm, maestro)
read(rn, novedades)

mientras no fin maestro && no fin novedades:
        si rm.id == rn.id:
                si rn.code == ‘A’:
                        write(rn, error)
                        write(rm, maestro_act)
                else si rn.code == ‘M’:
                        rm.desc = rn.desc
                        rm.valor = rn.valor
                        write(rm, maestro_act)
                read(rm, maestro)
                read(rn, novedades)
        else si rm.id > rn.id:
                si rn.code == ‘A’:
                        write(rn, maestro_act)
                si no:
                        write(rn, errores)
                read(rn, novedades)
        else:
                write(rm, maestro_act)
                read(rm, maestro)

mientras que no fin de maestro:
        read(rm, maesrtro)
        write(rm, maestro_act)

mientras que no fin de novedades:
        read(rn, novedades)
        rm.id = rn.id
        rm.desc = rn.desc
        rm.valor = rn.valor
        write(rm, maestro_act)

close maestro
close novedades
close maestro_act
close errores

remove maestro
rename (maestro_act, maestro)
```

9) Corte de Control: Del enunciado 2 realizar el informe de ventas por local agrupados por provincia y por ciudad.

```
open archivo ventas
read(reg, ventas)
mientras no fin ventas:
        provincia = reg.cod_prob
        print (“provincia ”, provincia)
        mientras no fin ventas && provincia == reg.cod_prob:
                ciudad = reg.cod_ciu
                print (“ciudad”, ciudad)
                mientras no fin ventas && provincia == reg.cod_prob && ciudad == reg.cod_ciu:
                        local = rec.cod_loc
                        ventas_loc = 0
                        mientras no fin ventas && provincia == reg.cod_prob && ciudad == reg.cod_ciu && local == reg.cod_loc:
                                ventas_loc += reg.total_rg
                                read(reg, ventas)
                        print(“local:”, local, “total:”, ventas_loc)
close ventas
```

10) Corte de Control: Del enunciado 1 realizar el informe de promedio de notas, agrupado por alumnos y facultad.

```
open archivo notas
read(reg, notas)
mientras no fin notas:
        suma_facu = 0
        cant_facu = 0
        facultad = reg.cod_fac
        print(“facultad”, facultad)
        mientras no fin notas && facultad == reg.cod_fac:
                suma_alu = 0
                cant_alu = 0
                alumno = reg.legajo
                mientras no fin notas && facultad == reg.cod_fac && alumno == reg.legajo:
                        suma_alu += reg.nota
                        cant_alu += 1
                        read(reg, notas)
                print(“Alumno: “, alumno, “promedio: “, suma_alu / cant_alu)
                suma_fac += suma_alu
                cant_facu += cant_alu
        print(“facultad”, facultad, “promedio:”, suma_facu / cant_facu)
close notas
```

11) Genere un archivo con más de 100.000 registros con la siguiente estructura: ID_Producto, Producto y Descripción. Ordenarlo con MergeSort. Entregar una breve descripción de cómo se generó el archivo, cuánto tardó el ordenamiento y si se presentaron dificultades. Utilizar distintas cantidades de memoria RAM y comparar los resultados.

El archivo fue generado con la estructura dada y con mas de 100.000 registros. Para la creación del mismo se utilizó un generador de números enteros random para generar los ID desordenados, y luego para realizar el ordenamiento se debía pasar los registros del archivo a un arreglo para poder ordenarlos por este método. El ordenamiento tardó aproximadamente 7,22 segundos para una cantidad de registros de 150.000 totalmente desordenados. Se implemento en Python, por lo que se utilizo una librería propia del lenguaje llamada time que tiene un método que ayuda a calcular el tiempo de ejecución entre una parte del código y otra. Luego de esto, los registros guardados en el arreglo ya ordenados se volvieron a guardar en el archivo, dejándolo completamente ordenado de forma ascendente por ID.

12) Busque al menos otros tres métodos de ordenamientos para archivos. Descríbalos y arme el pseudocódigo de los mismos.

Algoritmo: Selection Sort
Metodo: Seleccion
Complejidad: $O(n^2)$
Implementacion: C

```
void seleccion(int vec[], int n) {
    int min, i, k, aux;

    for (k = 0; k < n - 1; k++) {
        min = k;
        for (i = k + 1; i < n; i++)
            if (vec[i] < vec[min])
                min = i;
        if (min ≠ k) {
            aux = vec[k];
            vec[k] = vec[min];
            vec[min] = aux;
        }
    }
}
```

Algoritmo: Bubble Sort
Metodo: Intercambio
Complejidad: $O(n^2)$
Implementacion: C

```
void burbujeo(int vec[], int n) {
    int i, k, tope, aux;

    tope = n - 1;
    do {
        k = 0;
        for (i = 0; i < tope; i++)
            if (vec[i] > vec[i + 1]) {
                aux = vec[i];
                vec[i] = vec[i + 1];
                vec[i + 1] = aux;
                k = i;
            }
        tope = k;
    } while (k > 0);
}
```

Algoritmo: Shell Sort
Metodo: Insercion
Complejidad: $O(n(\log(n))^2)$
Implementacion: C

```
void shell(int vec[], int n) {
    int cambio, aux, i, paso;

    paso = n / 2;
    do {
        do {
            cambio = 0;
            for (i = 0; i < n - paso; i++) {
                if (vec[i] > vec[i + paso]) {
                    aux = vec[i];
                    vec[i] = vec[i + paso];
                    vec[i + paso] = aux;
                    cambio = 1;
                }
            }
        } while (cambio);
        paso = paso / 2;
        printf("Cambio de paso a %d\n", paso);
    } while (paso ≠ 0);
}
```

Algoritmo: Quick Sort
Metodo: Dividir
Complejidad: $O(n^2)$
Implementacion: C

```
void quickSort(int vec[], int pri, int ult, int n) {
    int i, j, aux, p;

    i = pri;
    j = ult;
    p = vec[(pri + ult) / 2];
    do {
        while (i ⩽ j && vec[i] < p)
            i += 1;
        while (i ⩽ j && vec[j] > p)
            j -= 1;
        if (i < j) {
            aux = vec[i];
            vec[i] = vec[j];
            vec[j] = aux;
        }
        i += 1;
        j -= 1;
    } while (i < j);
    if (pri < j)
        quickSort(vec, pri, j + 1, n);
    if (i < ult)
        quickSort(vec, i - 1, ult, n);
}
```