**Trabajo Práctico #1 : Metodología-DP-ECP-Módulos-ED: Registros y Arreglos.**

Se requiere de un proceso que informe las **diversos listados** de los corredores de un rally por cada etapa realizada y por cada tipo de vehículo. Para ello se cuenta con un archivo de datos “**EtapasRally.txt**” (texto) desordenado conteniendo:

**EtapasRally.txt**: a: Número de Etapa (1 díg.), b: Tipo de Vehículo (1 díg.), c: Número de vehículo (2 dígs.), d: Nombre del Piloto (str20), e: Nombre del CoPiloto (str20), f: Marca del vehículo (str15), g: Tiempo en minutos (3 dígs.). La cantidad máxima de registros está indicada por la constante **MAX\_REGS = 500.** Cantidad de **Etapas cinco (5)**, cantidad de **Tipos de Vehículos cuatro (4)**. El número de vehículo está asociado a un único corredor y solo se repite si corresponde al mismo corredor. **Máxima cantidad de vehículos cien (100).** Si un vehículo **abandona la carrera, el tiempo en minutos será de novescientos noventa y nueve (999)**, y en las próximas etapas restantes también por cada una de ellas el tiempo será de (999). Se deberán utilizar **constantes con nombres** para establecer los valores indicados anteriormente. Cada uno de los registros leídos del archivo EtapasRally.txt **deberán ser volcados a una estructura de datos interna**. Se podrán utilizar otras estructuras de datos adicionales para realizar determinados cálculos acumulativos de los tiempos obtenidos de cada etapa por cada número de vehículo. Se deberán determinar en algún momento del proceso las cantidades de: **vehículos asociados a un corredor sin repetir,** **vehículos que abandonaron la carrera** y **registros leídos** del archivo EtapasRally.txt.

Se cuenta con el bloque principal, solo se establecen las invocaciones a los módulos, ud. deberá declarar las variables pertenecientes a este módulo, reemplazar los nombres indicados en los argumentos por nombres de identificadores representativos o con significado para su propósito:

main() {

//declare ud. las variables de esta función y su inicialización, si corresponde.

Abrir (EtapasRally); // Dentro del módulo **ProcEtapasRally**, invocar a **LeeretapsRally**.

// Dentro de los módulos de los listados si corresponde

ProcEtapasRally(argumentos…); // se deberá invocar al módulo de OrdxBur.

ListadoCorredores(argumentos…); // En el módulo **ListadoCorredores** debe existir un

ListadoLargada(argumentos…); // freopen(nomFisArchivo, modo, stdout);

ListadoPuestosFinal(argumentos…); // En el módulo **ListadoGanadoresxTipoVehic** debe

ListadoGanadoresxTipoVehic(argumentos…); // volverse el freopen a la consola.

Cerrar (EtapasRally);

**return** 0;

} //main

**A continuación se aportan ideas para realizar el Algoritmo y su codificación:**

1. La función ***bool* *LeerEtapasRally(argumentos…)***, el cual lee una componente de datos del archivo y los almacena en una estructura interna. La función retorna un boolean, verdadero, si la lectura fue exitosa, caso contrario, falso.
2. La función ***void ListadoCorredores(argumentos…)***; debe salir en forma paralela a los registros leídos del archivo EtapasRally, recordando como se indicó más arriba desordenado, **a partir de los datos volcados a una estructura interna de datos**. El diseño es:

**Listado de Corredores de Etapas y Tipos de Vehículos**

**Etp. TpVehic. Nombre del Piloto Nombre del CoPiloto Marca Vehic. Tiempo mins.**

9 9 X(20) X(20) X(15) 999

1. La función ***void ListadoLargada(argumentos…);*** el cuál debe salir ordenado en forma ascendente por NroEtapa + TipoVehículo. El nro. de etapa debe salir solo una vez para cada cambio de nro. etapa.

**Listado de Largada**

**Nro. Etapa: 9**

**Tipo Vehículo: 9**

**Nro. Vehic. Nom. Piloto Nom. CoPiloto Marca Vehic.**

**99 X(20) X(20) X(15)**

. . . .

1. La función ***void*** **ListadoPuestosFinal(argumentos…)**, debe salir ordenado ascendente por Tiempo Total (en Minutos).

**Listado de Puestos Finales Carrera Rally**

**Puesto T.Vehic. Nro.Vehic. Nom. Piloto Nom. CoPiloto Marca Vehic. Tiempo Total**

**1 9 99 X(20) X(20) X(15) 999**

**2 . . . . . .**

**3 . . . . . .**

. . . . . . .

1. La función ***void* ListadoGanadoresxTipoVehic(argumentos…);** debe salir ordenado por tipo de vehículo ascendente.

**Listados de Ganadores por tipos de Vehiculos**

**Tipo Vehic. Nro. Vehic. Nombre del Piloto Tiempo Total**

1 99 X(20) 999

2 . . .

3 . . .

. . . .

**Cantidad de abandonos:** 99

Los listados serán destinados a un archivo en disco, con el nombre de “**ListadoRally.Txt**”, formateado con títulos y con los resultados obtenidos. Hay espacio suficiente para generar estos archivos.

**Observaciones, restricciones y recursos disponibles:**

Utilizar las siguientes funciones, invocando en donde sea necesario:

* ***void OrdxBur(tblEtapasTipos vE, short card);***
* ***void OrdxBur(short card, tblEtapasTipos vE);***
* ***void IntCmb(tipoX elem1, tipoX elem2); // Para cada tipo de ordenamiento y/o tbl***
* ***bool LeerEtapaRally(argumentos… ); // se realizará ajustado a c/u. de esos casos.***
* ***short BusBinVec(tblEtapasTipos vrE, short nVehic, short ult);***
* ***sTiempoTotal BusSecMinTmpxTipoV(tblTiemposTotales vrT, short tipoVehic, short cVehic);***
* ***void OrdxBur(tblTiemposTotales vT, short card); // TipoX representa un tipo de dato a pasar.***

**Espacio en disco**: Solo para generar los archivos de los listados en formato texto.

**Espacio para arreglos**: Lo necesario que requiera este proyecto.

**Espacio en memoria dinámica**: 0 bytes.

**Accesos a los archivos**: un solo recorrido secuencial, para leer a cada uno de ellos.

**Bloque Principal**: sólo invocaciones a módulos, según lo establecido anteriormente.

**Optimización**: dado que el uso de ciclos afecta el tiempo de ejecución de un proceso, se evaluará la eficiencia en el uso de los mismos.

Utilizar nombres significativos para los identificadores, dibujos para las estructuras de datos a utilizar, rotulando cada elemento, tamaño, breve leyenda de cómo se generan y estado inicial, respetar esos nombres para utilizarlos en el algoritmo. Preparar una muestra de datos para el archivo de datos e imprimirla, para ejecutar el programa. En el disco solo contendrán los archivos “**TP1V\_K1\_ \_ \_G\_-Apellido Nombre.cpp**”, y los archivos de datos “**EtapasRally.Txt**” (*de texto*), y el archivo de texto de la salida de los listados “**ListadoRally.Txt**” ubicados en el directorio raíz de la unidad.

* **Cada grupo debe crear su propia muestra de datos para el archivo.**

La cantidad máxima de registros está establecida al inicio de este documento.

Se deben utilizar constantes con nombres para indicar estas cantidades.

La muestra de datos para EtapasRally.Txt debe ser: 5 etapas, 4 tipos de vehículos y hasta un máximo de 25 vehículos de cada etapa-tipo, además debe registrarse por cada renglón los datos de un registro de un corredor, sin títulos, solo debe tener los datos, recordando que un mismo corredor puede estar como máximo 5 veces una por cada etapa, items indicados en la sección de archivos.

El Trabajo Práctico deberá ser entregado de acuerdo a las pautas indicadas más abajo, el cual se aprobará si reúne los requerimientos solicitados en **tiempo y forma**:

(**Se debe respetar el orden indicado a continuación**)

1. Entregar en carpeta tamaño A4 de tapa transparente y con sujetador de gancho para las hojas lo siguiente:
2. Carátula con los datos de los integrantes del TP1, entre 7 y 10 alumnos por grupo del mismo curso.
3. Esta misma hoja que establece el enunciado del problema a resolver. Cada grupo elegirá un líder del proyecto, que será el responsable de realizar las entregas del TP. Si un líder abandona la cursada, se deberá elegir otro líder.
4. Diseñar las estructuras de Datos graficándolas indicando con rótulos apropiados, cada elemento, su tamaño en bytes y las variables utilizadas. Las estructuras de datos a graficar son:
   1. *sEtapaRally*, el cual corresponde a cada uno de los registros de datos a leer del archivo de EtapasRally.
   2. *tblEtapasTipos*, el cual corresponde a la **estructura de datos combinada** que permite procesar los datos para poder emitir posteriormente los listados.
   3. *sTiempoTotal, el cual corresponde a una estructura de registro con los campos necesarios para poder procesar los tiempos totales en minutos.*
   4. *tblTiemposTotales*, el cual corresponde a otra **estructura de datos combinada** que permite procesar los cálculos de acumulación de tiempos totales en minutos.
5. Graficar el Bloque Principal.
6. Graficar cada uno de los módulos –funciones- a utilizar, cabecera y cuerpo.
7. Construir una muestra de datos, para el archivo de datos, la cual se la utilizará para probar el Algoritmo. En la hoja impresa a entregar debe haber rótulos apropiados, pero, NO en los archivos de Datos, “EtapasRally.txt”, el cual contendrán solamente, los datos. Ver detalle del formato más abajo.
8. Emitir según la muestra establecida, los resultados esperados, siempre acompañada de los rótulos apropiados.
9. Codificación del Algoritmo completo en el Lenguaje C++, emitiendo números de líneas. Usar Code-Blocks. Las primeras líneas serán de comentario indicando: Nombre del programa, fecha entrega, Nro. versión, breve comentario del objetivo del programa, datos del curso, nombre del día, turno, nro. del grupo e integrantes (Apellido, Nombre).

**Formato del archivo de datos EtapasRally.Txt**

Se deberá ajustar la escritura de los datos respetando a raja tabla este formato en donde cada columna representa un dato y su tipo de dato y ancho:

9 9 99 X(20) X(20) X(15) 999

Cada columna representa de izquierda a derecha lo siguiente:

**Nro.Etapa Tipo Vehic. Nro. Vehic. Nombre del Piloto Nombre del Copiloto Marca Vehic. Tiempo Mins.**

(1 díg.) (1 díg.) (2 dígs.) (20 caracteres) (20 caracteres) (15 caracteres) (3 dígs.)

Máx. Máx. Máx. Máx. Máx.

**Ejemplos:**

3 2 41 Juan Perez Pedro Picapiedra Ford 215

1 4 7 Ricardo Corazon Leon Ma Belle de Carlomag Alfa Romeo 93

4 3 25 Patricio Gomez Susana Torio VolksWagen 182

Cada nueva entrega además del nombre indicado para el archivo del código en C++ irá acompañado de la versión entregada, iniciando la primera entrega con el sufijo V1, luego la segunda entrega V2, y así sucesivamente. (S.E. ú O.).