En los apuntes se ve cómo programar una función de ordenamiento genérico, es decir que permite ordenar objetos de distinta naturaleza. Esto se logra haciendo que la función de ordenamiento reciba un puntero a una función que permite comparar los objetos. El problema de este ordenamiento es que exige que se ordenen arreglos de punteros. Si bien todavía es posible ordenar arreglos de números, el resultado no es completamente portable de una plataforma a otra.

Parte a.- Programe una función de ordenamiento completamente genérica. El encabezado debe ser:

```
typedef int (*Comparator)(void *ptr, int i, int j);
typedef void (*Swapper)(void *ptr, int i, int j);
void sort(void *ptr, int from, int to,
          Comparator compare, Swapper swap);
```

La función *sort* recibe un puntero opaco *ptr* hacia un objeto que representa una colección de elementos de tipo desconocido. Se accede a estos elementos con índices que van desde *from* hasta *to*. Además recibe un puntero a una función *compare* que compara los elementos con índices *i* y *j* de la colección y otra función *swap* que los intercambia.

El siguiente es un ejemplo de uso para ordenar un arreglo de números reales:

```
int compare_nums(void *ptr,
                 int i, int j) {
  double *nums= ptr;
  return nums[i]==nums[j] ? 0 :
         nums[i]<nums[j]  ? -1 :
                            1;
}
```

```
void swap_nums(void *ptr,
               int i, int j) {
  double *nums= ptr, aux;
  aux= nums[i];
  nums[i]= nums[j];
  nums[j]= aux;
}
```

```
double nums[]= { … };
sort(nums, 0, 9, compare_nums, swap_nums);
```

Ud. encontrará más ejemplos de uso en el archivo test-sort.c suministrado.

Restricción: Ud. no puede usar quicksort como algoritmo de ordenamiento ya que este se encuentra en los apuntes. Pero puede usar cualquier otro algoritmo. No se exige eficiencia.

Parte b.- En el programa de prueba se ordena un archivo de 10 líneas de 14 caracteres cada una, usando la función *sort*. Las primeras 10 columnas en el archivo corresponden al nombre de una persona. El ordenamiento requiere el comparador *compare_file_lines* y el intercambiador *swap_file_lines*. Programe estas 2 funciones usando la función de acceso directo *fseek*. El archivo debe quedar ordenado por el campo nombre. Los encabezados son:

```
int compare_file_lines(void *ptr, int i, int j);
void swap_file_lines(void *ptr, int i, int j);
```

El programa de prueba ordena el archivo usando este código:

```
FILE *file= fopen("people-sort.txt", "r+");
sort(file, 0, 9, compare_file_lines, swap_file_lines);
```

Observe que el archivo es de lectura/escritura. Ud. necesitará leer y reescribir las líneas *i* y *j* en *swap_file_lines*.

Restricción: Ud. no puede leer el archivo completo en memoria. A lo más se le permite leer y reescribir 2 líneas del archivo en cada llamada de estas funciones[1].

Recursos

Baje del material docente de U-cursos el archivo *t2.zip*. Descomprímalo y encontrará los siguientes archivos:

- *Makefile*: permite compilar, ejecutar y validar que su tarea funciona correctamente.
- *sort.h*: Archivo con los encabezados de las 3 funciones pedidas y los tipos requeridos.
- *test-sort.c*: El programa de prueba.
- *out.txt*: Lo que el programa de prueba debe entregar por la salida estándar.
- *people.txt*: el archivo original que ordena el programa de prueba. Este ordena una copia en *people-sort.txt*.
- *run-test*: script que valida que el programa de prueba entregue resultados correctos. Se invoca desde Makefile.

Ud. debe crear un archivo *sort.c* en donde debe programar las funciones *sort*, *compare_file_lines* y *swap_file_lines*. Luego compile y pruebe su tarea ejecutando:

```
$ make
```

Se le felicitará si las funciones trabajan correctamente. De otro modo, el comando de Unix *diff* le indicará las diferencias entre lo entregado y lo requerido en la salida estándar.

Entrega

Entregue en U-cursos el archivo *sort.c* en donde Ud. programó las 3 funciones pedidas. No incluya archivos binarios. El plazo de entrega de la tarea vence el día Martes 5 de Mayo. Se descontará medio punto por día de atraso (excepto Sábados, Domingos y festivos).

1 Esta no es una forma eficiente de ordenar un archivo, pero es pedagógicamente útil para demostrar el uso de punteros a funciones y el acceso directo en archivos (*fseek*).