Ejercicio 1:

Escriba un programa que solicite al usuario el ingreso de números enteros y los guarde en un archivo de acuerdo a los siguientes requerimientos:
-Se solicitará el ingreso de números hasta que se ingrese un valor igual a 0 que no debe ser guardado.
-Al finalizar el ingreso se solicitará ingresar el nombre del archivo en el que se desean guardar los números.
-El formato a utilizar en el archivo será de cadenas de 10 caracteres. Para los números que tengan menos de 10 caracteres se rellenará con el carácter '0' hasta completar la cadena. Los cadenas deben quedar separadas en el archivo mediante un salto de línea CRLF.
-Se informará si el archivo fue guardado con éxito o si falló y luego terminará la ejecución del programa.

Ejercicio 2:

Se desea realiza un software que simule una transacción financiera. El mismo deberá solicitar un monto, número de tarjeta y código de seguridad por teclado. Luego enviará un mensaje a un host que devolverá el estado de la transacción (aprobada o rechazada).
-Solicitar el monto de la compra con 2 decimales para los centavos
-Solicitar el número de tarjeta (longitud variable, mínimo 13 dígitos)
-Verificar que el número de tarjeta corresponda a una tarjeta válida (ver "Reconocimiento de tarjetas"). Si no es válido mostrar mensaje "TARJETA NO SOPORTADA" en pantalla y abortar la operación, de lo contrario mostrar el *label* de la tarjeta en pantalla y pasar al siguiente paso.
-Solicitar el código de seguridad (3 dígitos).
-Armar el "request message" con los datos de la transacción (ver "Formato de los mensajes") y enviarlo al host.
-Esperar y leer el "response message" (ver "Formato del mensaje"). Si transcurren más de 5 segundos o si ocurre otro error, deberá mostrarse en pantalla "ERROR DE COMUNICACION" y abortar el proceso.
-Mostrar la respuesta en pantalla. Si el código de respuesta es "00", indica que la transacción fue aprobada y deberá mostrar "APROBADA" en pantalla. Si el código de respuesta es cualquier otro valor, deberá mostrar "RECHAZADA".

NOTA: para el envío y recepción de mensajes puede utilizar las siguientes funciones:

```
/**
* Crea un socket
* @return handle del socket o -1 en caso de error
*/
int socketCreate();

/**
* Conectar el socket a un host
* @param handle handle del socket a utilizar
* @param ip direccion ip del host a conectarse
* @param port puerto del host a conectarse
* @return 0 en caso de exito o -1 en caso de error
*/
int socketConnect(int handle, const char *ip, unsigned short port);

/**
* Leer datos del socket. La función bloquea hasta leer datos o transcurrir maxTimeout
milisegundos
* @param handle handle del socket a utilizar
```

```
* @param data puntero donde se guardarán los datos leidos
* @param maxTimeout tiempo máximo de espera en milisegundos
* @return cantidad de bytes leidos o -1 en caso de error
*/
int socketRead(int handle, unsigned char *data, int maxTimeout);

/**
* Escribir datos en un socket
* @param handle handle del socket a utilizar
* @param data puntero desde donde se leerán los datos a escribir
* @return cantidad de bytes escritos o -1 en caso de error
*/
int socketWrite(int handle, const unsigned char *data);

/**
* Cierra el socket y libera recursos
* @param handle handle del socket a utilizar
* @return 0 en caso de exito o -1 en caso de error
*/
int socketClose(int handle);
```

Reconocimiento de tarjetas:

Se utilizarán 2 archivos con registros: ranges.dat y cards.dat. Cada uno de ellos está compuesto por registros guardados con el formato dado por las siguientes estructuras:

```
typedef struct {
	char rangeLow[8 + 1];
	char rangeHigh[8 + 1];
	unsigned char len;
	int id;
} range_t;

typedef struct {
	char label[12 + 1];
	int id;
} card_t;
```

El proceso es el siguiente:

Dado un número de tarjeta, se leerán uno por uno los registros del archivo ranges.dat cuyo formato está dato por la estructura *range_t*. Para cada registro se deberá verificar que los 8 primeros dígitos del número de tarjeta estén incluidos dentro del rango, y que además la longitud del número de tarjeta coincida con *len*.
El rango está dado por *rangeLow*, que indica el límite inferior del rango y *rangeHigh*, que indica el límite superior. Los 8 primeros dígitos de tarjeta deben estar comprendidos entre ambos (o ser iguales a alguno de dichos extremos).
Si estas condiciones no se cumplen, se siguen leyendo los registros del archivo hasta llegar al final del mismo o hasta encontrar un registro en el que se cumplan. Si las condiciones se cumplen, se considera que el número de tarjeta pertenece a dicho rango y se deberá utilizar el elemento *id* de la estructura para identificar el tipo de tarjeta en el archivo cards.dat.
El archivo cards.dat contiene registros con formato dado por la estructura *card_t*. Cada uno de ellos contiene un *id* único dentro del archivo que permite identificar cada registro y asociarlo con los rangos del archivo ranges.dat. El label es un string que contiene el nombre de la tarjeta a ser mostrado en pantalla.

Formato de los mensajes:

Todos los campos de los mensajes tienen formato ASCII.

Request message:

Tipo de mensaje	Número de tarjeta	Monto	Código de seguridad

Tipo de mensaje: 0200
Número de tarjeta: se indicará con 2 caracteres la longitud y a continuación el número de tarjeta
Monto: 12 caracteres. Rellenar con 0 a la izquierda y no poner separador de centavos. Ejemplo: $12,53 se enviaría como “000000001253”
Código de seguridad: 3 dígitos

Response message:

Tipo de mensaje	Código de respuesta

Tipo de mensaje: 0210
Código de respuesta: 2 dígitos