Apellido, Nombre:

.....

Mail:

.....

Padrón:

.....

Teórico / Práctico - Entrego ..... hojas				
1:	2:	3:	4:	Nota: .....

RPL		
1:	2:	Nota: .....

Nota Final
.....

Aclaraciones:

- **Antes de comenzar a resolver el parcial, complete sus datos en esta hoja, y al finalizarlo, firme todas las hojas.**
- **Los ejercicios deben ser implementados en el lenguaje de programación C, respetando las buenas prácticas de programación.**
- **Para cada ejercicio práctico se recomienda fuertemente realizar un análisis y un diagrama del problema y la solución.**
- **Se deben numerar TODAS las hojas e inicializarlas con nombre, apellido, padrón y cualquier otra información que considere necesaria.**
- **La aprobación del parcial está sujeta a la aprobación de los 2 módulos del mismo (teoría/práctica en papel y práctica en RPL).**

Ejercicios:

1. **Fawkes** es un fénix muy inteligente, mascota y defensor de **Albus Dumbledore**. Tiene la capacidad de, cuando cree que la hora de partir ha llegado, prenderse en llamas y renacer de sus cenizas.
   Albus creó el siguiente procedimiento que cuenta cuántos renacimientos lleva **Fawkes**.

```
void renacer_de_las_cenizas(bool es_hora, int cantidad_renacimientos){
      if (es_hora)
            cantidad_renacimientos++;
}
```

   Sin embargo, **Dumbledore** afirma que **anda mal**.
   a. ¿Está **Dumbledore** en lo cierto? ¿Qué cambiaría para que funcione correctamente?
   b. Sabiendo que a lo largo de su vida **Fawkes** toma 3 colores: *Rojo* (al nacer), *Amarillo* (al crecer) y *Carmesí* (en su última etapa), ¿qué tipo de dato usaría para representar estas etapas y que convención usaría para ellas? Explique por qué la eligió.

2. **Garrick Ollivander** está haciendo su limpieza semanal de varitas. Para esto, las toma todas y las limpia una a una. El siguiente algoritmo recursivo, muestra el accionar de **Garrick**.

```
void limpiar_varitas_de_la_tienda_de_garrick_recursivamente(varita_t
v[MAX_VARITAS], int t, int actual){
if (actual < t){
limpiar_actual_varita(v[actual]);
limpiar_varitas_de_la_tienda_de_garrick_recursivamente(v,tope,actual+1);
}
}
```

   Asumiendo que la rutina recursiva funciona correctamente y que el procedimiento **limpiar_actual_varita** está implementado:
   a. ¿Qué buenas prácticas se están incumpliendo? ¿Qué cambios haría para cumplirlas?
   b. ¿Cuáles son los componentes esenciales en una rutina recursiva?.

"Al fin y al cabo, somos lo que hacemos para cambiar lo que somos" - Eduardo Galeano.

3. **La Sala de las Profecías** es una cámara en el **Departamento de Misterios**, en el nivel nueve del **Ministerio de Magia**. La misma, cuenta con estantes que contienen los registros de las profecías, aunque muchos de éstos, si no todos, fueron destruidos cuando una parte de una batalla entre mortífagos y los miembros del **Ejército de Dumbledore** se produjo en el salón. Cada esfera de cristal contiene el registro de una profecía y se etiqueta con el nombre del profeta, la persona a quien se refiere, y el año en que se hizo la profecía.

```
typedef struct profecia{
	char profeta[MAX_PROFETA];
	char destinatario[MAX_DESTINATARIO];
	int anio;
} profecia_t;
```

   a. **Harry Potter** entró a **La Sala de Profecías** a buscar aquella que lo tiene como destinatario y fue originada por **Sybill Trelawney.** Crear una función que reciba como parámetro un vector de profecías y su tope y devuelva como resultado de la función la posición de la buscada por **Harry**, sabiendo que el vector no tiene ningún orden particular y que solo existe una profecía que tiene como profeta a **Sybill Trelawney** y como destinatario a **Harry Potter.**
   b. Explique un método de ordenamiento. No debe escribir el algoritmo, solo explicarlo.
   c. ¿En este caso en particular, convendría ordenar las profecías? ¿Por qué?

4. Pasó el examen *multiple choice* de **Dolores Umbridge**. Harry y Ron salieron aterrados por la dificultad del mismo y lo primero que quieren hacer es comparar sus respuestas.
El examen constaba de 100 preguntas pero al ser tan difícil, no pudieron responderlas todas. Ellos quieren saber cuántas preguntas respondieron igual, para tener una noción de cómo les fue.
Se tiene el siguiente registro de una respuesta.

```
typedef struct respuesta{
	int id_pregunta;
	char respuesta;
} respuesta_t;
```

   Se requiere una función que reciba como parámetro los vectores de respuestas de **Harry** y **Ron** y sus topes, y devuelva como resultado la cantidad de preguntas que contestaron igual. Recuerde que dichos vectores no tienen necesariamente la misma cantidad de elementos ya que si no contestaron la pregunta, esta no aparecerá en el vector de respuestas.

   *Ambos vectores están ordenados por **id_pregunta**.*

   a. Realizar la función que resuelva el problema.
   b. ¿Cuánto ocupa en memoria el vector de respuestas de **Harry** para una arquitectura de 64 bits? ¿Y el de **Ron**?
   c. Si **Ron** supiera que una de las preguntas que no respondió es **superhipermegaarchi** importante y tiene la posibilidad de agregarla, ¿cómo lo haría?. Recuerde que el vector debe quedar ordenado. No debe escribir el algoritmo, solo explicarlo.

//TEORICO-PRACTICO//
/
1a) Dumbledore está en lo cierto.

-El dato "int cantidad_renacimientos" debería ser pasado por referencia a la función, así cuando este se use como contador la información se guardara correctamente.

1b) Se podría usar tipo de dato char, y definir las siguientes constantes:

```
#define COLOR_AL_NACER 'R'
#define COLOR_CRECIMIENTO 'A'
#define COLOR_ULTIMA_ETAPA 'C'
```

Elegí char ya que puedo representar cada color de cada etapa con su inicial.

2a) -Indentación: El código se debería indentar dandole espacio entre el marco izquierdo y el "if", luego doble espacio a cada acción dentro de la función "if".
-Variable mal nombrada: La variable "int t" que recibe la función tiene un nombre muy genérico y se puede llegar a confundir el propósito de dicha variable. Un nombre adecuado sería "int tope" ya que es su objetivo en la función.
-Función mal nombrada: Nombre muy extenso y con detalles innecesarios. Un ejemplo: "void limpiar_varitas(...".
-Vector mal nombrado: Al igual que la variable, "v[]" es un nombre muy genérico. Un ejemplo: "varitas[]", este da a entender que es un vector de varitas.

2b) En una rutina recursiva se debe lograr un buen uso del código, evitando el uso de funciones iterativas, para lograr una función o un procedimiento en el cual repita la misma acción para distintos elementos cumpliendo un ciclo. Es esencial el corte adecuado a la hora de programar, ya que sino la rutina se repetiria infinitamente, llamandose la así misma indefinidamente. Componentes: Condicion de corte y llamada recursiva.

3a) Análisis:
Debemos crear una función que reciba como parámetros un vector de profecías (desordenado) con su tope, y que devuelva como resultado de la función la posición de la profecía buscada por Harry, sabiendo que es única.
Primero, necesitaremos declarar una variable de tipo entero para guardar la posición de la profecía que buscamos. Luego recorreremos el vector en su totalidad buscando dicha profecía, comparando los strings de los nombres tanto del profeta como del destinatario con los que tiene Harry (Profeta: Sybill Trelawney / Destinatario: Harry Potter). Una vez que ambos nombres coincidan estaremos en la posición buscada y esta la guardaremos en la variable que declaramos al inicio.
Finalmente, una vez que vector llegue al final del recorrido, vamos a devolver como resultado de la función el valor que guardamos en la variable.

Código:

```
#include <stdio.h>
#include <string.h>

#define PROFETA_BUSCADO "Sybill Trelawney"
#define DESTINATARIO_BUSCADO "Harry Potter"
#define COINCIDENCIA 0

int buscar_profecia(profecia_t profecias[MAX_PROFECIAS], int tope){
  int comparacion_profeta;
  int comparacion_destinatario;
  int posicion_buscada;
```

```
  for(int i = 0; i < tope; i++){
    comparacion_profeta = strcmp(profecias.profeta, PROFETA_BUSCADO);
    comparacion_destinatario = strcmp(profecias.destinatario,
DESTINATARIO_BUSCADO);

    if( (comparacion_profeta == COINCIDENCIA) && (comparacion_destinatario ==
COINCIDENCIA) ){
      posicion_buscada = i;
    }
  }
  return posicion_buscada;
}
```

3b) METODOS DE ORDENAMIENTO

SELECCION
*Se busca el elemento de menor valor en el arreglo y se lo intercambia con el elemento que esta en la primera posición del arreglo.
*Posteriormente se busca el segundo elemento mas pequeño del arreglo y se lo intercambia con el elemento de la segunda posición del arreglo.
*Se continúa aí hasta que todos los elementos del arreglo están ordenados.

3c) No conviene ya que se va a realizar una sola búsqueda.

4a) Análisis:
Debemos crear una función que reciba como parámetro los vectores de respuestas de Harry y de Ron y sus topes, los cuales no son necesariamente iguales; y que
devuelva como resultado la cantidad de preguntas que contestaron igual. Sabemos que ambos vectores están ordenados por id_pregunta.
Primero necesitaremos declarar tres variables de tipo entero, dos contadores para las posiciones de los vectores, y un tercero para la cantidad de respuestas
iguales.
Luego recorreremos los vectores, comparadando el id_pregunta de ambos en cada posicion, dentro de un ciclo, el cual va a cortar cuando se llegue al
final de cualquiera de los dos vectores (comparando el valor de los contadores con los topes). Vamos a tener dos situaciones para analizar, y dependiendo de cada una
veremos cual contador aumentaremos:
A) Los id_pregunta de ambos vectores coinciden:
Aa) Las respuestas coinciden:
-Se aumenta el contador de respuestas iguales.
-Se aumenta el contador para la posición del vector de Harry.
-Se aumenta el contador para la posición del vector de Ron.
Ab) Las respuestas no coinciden:
-Se aumenta el contador para la posición del vector de Harry.
-Se aumenta el contador para la posición del vector de Ron.

B) Los id_pregunta de ambos vectores no coinciden:
Ba) Si el valor de la posición del contador del vector de Harry es mayor a el valor de la posición del contador del vector de Ron:
-Se aumenta el contador para la posición del vector de Ron.

Bb) Si el valor de la posición del contador del vector de Harry es menor a el valor de la posición del contador del vector de Ron:
-Se aumenta el contador para la posición del vector de Harry.

Una vez que el ciclo corte, obtendremos la cantidad de respuestas iguales que tuvieron Harry y Ron en su examen, la cual devolvemos como retorno de la

función pedida.

Código:

```
#include <stdio.h>

#define MAX_PREGUNTAS 100

int comparar_respuesta(respuesta_t harry[MAX_PREGUNTAS], respuesta_t
ron[MAX_PREGUNTAS], int tope_harry, int tope_ron){
  int respuestas_iguales = 0;
  int i = 0;
  int j = 0;

  while( (i < tope_harry) && (j < tope_ron) ){
    if(harry[i].id_pregunta == ron[j].id_pregunta){
      if(harry[i].respuesta == ron[j].respuesta)
        respuestas_iguales++;
      i++;
      j++;
    }
    else if(harry[i].id_pregunta > ron[j].id_pregunta)
      j++;
    else
      i++
  }
  return respuestas_iguales;
}
```

4b) Primero calculo cuanto ocupa el tipo de dato respuesta_t para una arquitectura de 64 bits:

int id_pregunta 4 bytes
char respuesta 1 byte

respuesta_t 5 bytes

respuesta_t harry[100] = respuesta_t ron[100] = 5 bytes * 100 = 500 bytes

4c) Usaría el método de inserción ordenada de un elemento dentro de un vector. Se recorre el vector de respuestas guardando la información del mismo en un nuevo vector ordenado, hasta llegar a la posición en donde se inserta la nueva respuesta. Luego se termina de recorrer el vector de respuestas guardando la información restante.