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. **Severus Snape** decidió crear un algoritmo que le diga la cantidad de alumnos varones y la cantidad de alumnos mujeres que hay en su clase. Acostumbrado a escribir código en el lenguaje de programación Python escribió el siguiente código en C:

```c
int calcular_cantidad_por_genero(char alumnos[TAM]){
    size_t cantidad_hombres;
    size_t cantidad_mujeres;
    for(int i = 0; i < TAM; i++) {
        if(alumnos[i] == 'F')
            cantidad_mujeres++;
        else
            cantidad_hombres++;
    }
    return cantidad_mujeres, cantidad_hombres;
}
```

   a. ¿Hay algo mal en este código? Explique por qué.
   b. ¿Cómo haría para resolverlo y que funcione? (No es necesario escribir código)

2. **Rubeus Hagrid** necesita una estructura para catalogar a los animales del bosque. Los quiere catalogar de acuerdo a la cantidad de patas, el color de su pelaje (verde, rojo, azul, violeta, amarillo, blanco u otro), su altura (en cm), peso (en kg), tipo de dieta (Carnívoro, Herbívoro, Insectívoro u Omnívoro).

   a. Definir un tipo de registro que le sirve a Hagrid para catalogar los animales.
   b. ¿Cuántos bytes ocupa en memoria el registro para una arquitectura de 64 bits?.

3. **Hermione** siempre fué muy competitiva, está vez, consiguió el registro de las notas de todos sus compañeros de las clases de *Defensa Contra las Artes Oscuras*, *Encantamientos*, *Pociones y herbología*. **Hermione**, quiere saber la nota de sus compañeros. Para ayudarla,

tenemos que crear un algoritmo que reciba un vector de alumnos (*alumno_t*), el nombre de un alumno y devuelva el promedio de notas. La firma de la función es la siguiente:

```
int buscar_alumno(alumno_t alumnos[MAX_ALUMNOS], char* nombre_alumno) {
	//...
}
```

```
typedef struct alumno {
	char nombre [MAX_NOMBRE];
	int edad;
	int defensa_artes_oscuras;
	int encantamientos;
	int pociones;
	int herbologia;
} alumno_t;
```

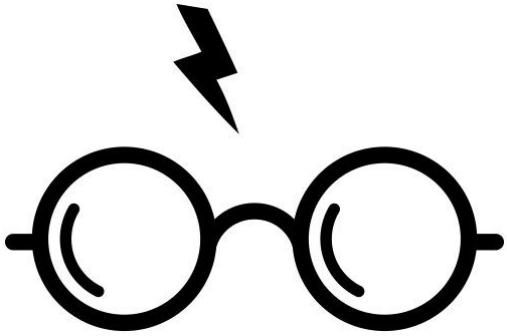
4. **Ron Weasley** está a cargo de la granja de zanahorias de su familia. Diariamente, tiene que recorrer la granja y revisar cada zanahoria. Realizar este trabajo es agotador, así que **Ron** nos pidió ayuda para crear un algoritmo que le diga si vale la pena recorrer la granja. Para **Ron**, vale la pena recorrerla cuando más del 40% de las zanahorias están maduras.
Además, tiene que tener un registro de la cantidad de zanahorias que les roban diariamente los conejos.
Se tiene que diseñar una función que cumpla con la siguiente firma:

```
bool recorrer_granja(char granja[MAX_GRANJA][MAX_GRANJA], int
ancho_granja, int *zanahorias_robadas){
    //...
}
```

El campo granja puede poseer los siguientes caracteres:

```
' ': La posicion está vacía.
'Z': Hay una zanahoria no madura.
'M': Hay una zanahoria madura.
'R': La zanahoria que estaba en esta posición fue robada.
```

La función debe devolver true o false dependiendo de si vale la pena o no recorrer la granja. Además, debe devolver a través del parámetro **zanahorias_robadas** la cantidad de zanahorias robadas por los conejos.
Notas:
- Las zanahorias robadas no cuentan a la hora de saber si vale la pena recorrer la granja o no.
- La granja es cuadrada.



"Se nace lo que se es. . . O se será aquello lo que se crea. . . " - Jorge Drexler.

RPL:

1. Hagrid tiene una huerta de calabazas, quiere saber cuanto dulce y cuanta sobra para su lumbricario puede sacar de ella.
   Cada calabaza tiene,
   Un estado:
   - Verde, no puede usarse para dulce.
   - Madura, puede usarse entera para dulce.
   - Pasada, solo puede usarse la mitad de la calabaza para dulce y la otra mitad para el lumbricario

   Un peso (en kilos):
   - De ese peso, la mitad es cáscara y semillas, que no pueden ser usados para dulce pero si para el lumbricario.

   Sabiendo que solo cosechará las maduras y pasadas, crear una función que devuelva la cantidad de calabazas verdes que quedan sin cosechar (como retorno de ella), la cantidad (en kilos) de calabaza para dulce (en un parámetro pasado por referencia) y la cantidad (en kilos) de calabaza para el lumbricario.

   *Aclaración: Todas las variables correspondientes a cantidades deben ser de tipo **int**.*

```
#define MAX_CALABAZAS 400
typedef struct calabaza{
    char estado; // V: verde - M: madura - P: pasada
    int peso;
} calabaza_t;
int hacer_dulce(calabaza_t calabazas[MAX_CALABAZAS], int tope_calabazas, int*
kilos_dulce, int* kilos_lumbricario){
    // Aqui va tu codigo...
}
```

2. *Harry* está caminando por *Hogwarts* a altas horas de la noche, cosa que está prohibido en la escuela. Los profesores escucharon ruidos extraños por los pasillos y están buscándolo.
   Por suerte, trae consigo el *Mapa del Merodeador* y puede saber que donde está cada uno de los profesores. El mapa puede representarse como una matriz, *Harry* está en la posición (0,0) y es representado por una **H**.
   En él vio que los que lo buscan son:
   - *Severus Snape* representado por una **S**.
   - *Remus Lupin* representado por una **L**.
   - *Alastor Moody* representado por una **M**.
   - *Dolores Umbridge* representada por una **U**.

   Si bien puede verlos, necesita saber cual está mas cerca para saber con quien tendrá que lidiar primero.
   Se pide realizar una función que devuelva el caracter que representa al profesor que se encuentra mas cerca.
   Como *Harry* está en la posición (0,0) de la matriz, la distancia puede calcularse como la suma de las coordenadas de la posición del profesor.
   En caso de haber 2 o más profesores a la misma distancia, devolver **X**.

```
#define MAX_MAPA 400
char profesor_mas_cercano(char mapa[MAX_MAPA][MAX_MAPA], int alto, int ancho){
    // Aqui va tu codigo...
}
```

"Se nace lo que se es. . . O se será aquello lo que se crea. . . " - Jorge Drexler.

//TEORICO-PRACTICO//

1a) -Las variables "cantidad_mujeres" y "cantidad_hombres" no estan inicializadas.
-La función devuelve erroneamente las dos variables antes mencionadas, cuando solo puede devolver un valor.

1b) -Haria de la función un procedimiento que reciba como parámetros por referencia las variables antes mencionadas, de modo que se puede devolver por referencia la
cantidad de alumnos en cada variable.

2a)

```
#define MAX_PELAJE 20
#define MAX_DIETA 20

typedef struct animal{
  int cant_patas;
  char color_pelaje[MAX_PELAJE];
  int altura_cm;
  int peso_kg;
  char dieta[MAX_DIETA];
} animal_t;
```

2b) Primero debo calcular cuanto ocupa cada tipo de dato dentro del struct en una arquitectura de 64 bits:

int cant_patas; = 4 bytes
char color_pelaje[MAX_PELAJE]; = 1 byte * 20 (MAX_PELAJE) = 20 bytes
int altura_cm; = 4 bytes
int peso_kg; = 4 bytes
char dieta[MAX_DIETA]; = 1 byte * 20 (MAX_DIETA) = 20 bytes

animal_t conejo_salvaje = 52 bytes

3) Análisis:
Para ayudar a Hermione, debemos crear un algoritmo que reciba un vector de alumnos, el nombre de un alumno y que devuelva el promedio de notas.
Primero definiremos e incializaremos en cero una varible de tipo entero (por firma de función) para luego almacenar el promedio calculado allí. Recorremos el
vector recibido comparando el nombre recibido con el nombre del alumno en cada posición del vector. En caso de haber coincidencia, sumamos la nota de Defensa contra
las Artes Oscuras con la nota de Encantamientos, con la de Pociones y con la de Herbología, y divimos esta suma por cuatro (cantidad de materias) obteniendo así el
promedio de dicho alumno.
Una vez finalizado el recorrido del vector, se devuelve como resultado de la función el promedio almacenado en la variable definida al comienzo.

Código:

```
#include <string.h>

#define COINCIDENCIA 0
#define NO_COINCIDENCIA 1
#define CANT_MATERIAS 4

int buscar_alumno(alumno_t alumnos[MAX_ALUMNOS], char* nombre_alumno){
  int comparar_nombre = NO_COINCIDENCIA;
  int promedio = 0;
  int i = 0;
```

```
  while( (comparar_nombre != COINCIDENCIA) || (i < MAX_ALUMNOS) ){
    comparar_nombre = strcmp(nombre_alumno, alumnos[i].nombre);

    if(comparar_nombre == COINCIDENCIA){
        promedio = (alumnos[i].defensa_artes_oscuras +
alumnos[i].encantamientos + alumnos[i].pociones + alumnos[i].herbologia) /
CANT_MATERIAS;
    }

    i++;
  }
  return promedio;
}
```

4) Análisis:

Debemos crear una función para recorrer la granja de la familia de Ron y revisar cada zanahoria para saber si tiene o noo que recorrerla. Vale la pena cuando
más del 40% de las zanahorias están maduras. Además, se tiene que tener un registro de la cantidad de zanahorias que les roban diariamente los conejos (se recibe por
referencia).

Primero debemos definir variables de tipo entero para usar como contadores, una para el total de zanahorias y otra para las zanahorias maduras, ambas
inicializadas en cero. Además de una variable del tipo numero de coma flotante para luego calcular el porcentaje de zanahorias maduras.

Recorremos la matriz recibida que representa la granja de la familia de Ron con un doble ciclo, verificando cada posición si no está vacía:

-Si la zanahoria está madura: Se aumenta el contador total de zanahorias y el contador de zanahorias maduras.

-Si la zanahoria no está madura: Se aumenta el contador total de zanahorias.

-Si hay señales de que la zanahoria fue robada: Se aumenta la variable de zanahoria robadas recibida por referencia.

Cuando se termina de recorrer la granja, se calcula el porcentaje de zanahorias maduras y si este es mayor al 40%, la función devuelve como resultado true,
en caso contrario devuelve false.

Código:

```
#define PORCENTAJE_RECORRER 40
#define VACIA ' '
#define MADURA 'M'
#define NO_MADURA 'Z'
#define ROBADA 'R'

bool recorrer_granja(char granja[MAX_GRANJA][MAX_GRANJA], int ancho_granja,
int* zanahorias_robadas){
  bool hacer_recorrido;
  int total_zanahorias = 0;
  int zanahorias_maduras = 0;
  float porcentaje_maduras;

  for(int i = 0; i < ancho_granja; i++){
    for(int j = 0; j < ancho_granja; j++){

      if(granja[i][j] != VACIA){

        if(granja[i][j] == MADURA){
                total_zanahorias++;
          zanahorias_maduras++;

        }
```

```
        else if(granja[i][j] == ROBADA){
          (*zanahorias_robadas)++;
        }
             else{
               total_zanahorias++;
             }
      }
    }
  }

  porcentaje_maduras = (zanahorias_maduras * 100) / total_zanahorias;

  if(porcentaje_maduras > PORCENTAJE_RECORRER){
    hacer_recorrido = true;
  }
  else{
    hacer_recorrido = false;
  }

  return hacer_recorrido;
}


////////////////////////////////////////////////////////////////////////////R
PL////////////////////////////////////////////////////////////////////////
1)Análisis:
          Debemos crear una función que devuelva la cantidad de calabazas
verdes que quedan sin cosechar (como retorno de ella), la cantidad (en  kilos)
de calabaza
   para dulce (en un parámetro pasado por referencia) y la cantidad (en
kilos) de calabaza para el lumbricario. Por la firma de la función, sabemos que
la huerta de
   Hagrid es representada por un vector.
          Primero debemos declarar una variable de tipo entero inicializada en
cero para contar las calabazas que queden sin cosechar, y luego hay que
recorrer el
   vector y verificar el estado de las calabazas:
               -Si está verde: Se aumenta el contador de calabazas sin
cosechar.
               -Si está madura: La mitad de su peso se le suma a la variable
de kilos para dulce recibida por referencia.
                                La otra mitad (cáscara y semillas) se le suman
a la variable de kilos para lumbricario también recibida por referencia.
               -Si está pasada: La mitad de la mitad de su peso, es decir un
cuarto, se le suma a la variable de kilos para dulce recibida por referencia.
                                El resto, tres cuartos de su peso, (cáscara,
semillas y partes pasadas) se le suman a la variable de kilos para lumbricario
                                también recibida por referencia.
          Una vez finalizado el recorrido del vector, se devuelve como retorno
de la función el valor de calabazas maduras en el contador de calabazas sin
cosechar.

Código:
#define MADURA 'M'
#define PASADA 'P'

typedef struct calabaza{
  char estado; // V: verde - M: madura - P: pasada
  int peso;
} calabaza_t;

int hacer_dulce(calabaza_t calabazas[MAX_CALABAZAS], int tope_calabazas, int*
kilos_dulce, int* kilos_lubricario]){
  int sin_cosechar = 0;
```

```
  for(int i = 0; i < tope_calabazas; i++){
    if(calabazas[i].estado == MADURA){
      *kilos_dulce += calabazas[i].peso / 2;
      *kilos_lubricario += calabazas[i].peso / 2;
    }

    else if(calabazas[i].estado == PASADA){
      *kilos_dulce += calabazas[i].peso / 4;
      *kilos_lubricario += calabazas[i].peso / 0.75;
    }
    else{
      sin_cosechar++;
    }
  }
  return sin_cosechar;
}
```

2)Análisis:

Se nos pide realizar una función que devuelva el caracter que representa al profesor que se encuentra más cerca de Harry. La distancia puede calcularse como la suma de las coordenadas de la posición del profesor, sabiendo que Harry está en la posición (0,0) y que el mapa de la escuela está representado por una matriz cuadrada.

Primero debemos definir una variable de tipo entero para la distancia, inicializada con el valor maximo de distancia posible, y otra de tipo caracter para almacenar la inicial del profesor que se encontrará más cerca. Luego recorremos la matriz, y en caso de encontrarnos con la posición de un profesor, se calcula la distancia del mismo con respecto a Harry. Se compara la distancia del encontrado con la anterior (si es el primero, con la max distancia inicializada al comienzo), y en caso de encontrarse más cerca, se guarda su inicial en la variable de tipo char. Si se encuentran a la misma distancia, es un "empate" y se guarda el caracter 'X'.

Una vez finalizado el recorrido de la matriz que representa el mapa, vamos a obtener en la variable de tipo char la inicial del profesor que se encuentra más cerca de Harry (o 'X' si son más de uno), y esta se devuelve por retorno de la función.

Código:

```
#define MAX_MAPA 400
#define MAX_DISTANCIA 800
#define SNAPE 'S'
#define LUPIN 'L'
#define MOODY 'M'
#define DOLORES_UMBRIDGE 'U'
#define VARIOS_CERCA 'X'

char profesor_mas_cercano(char mapa[MAX_MAPA][MAX_MAPA], int alto, int ancho){
  int distancia;
  int menor_distancia = MAX_DISTANCIA;
  char mas_cerca;

  for(int i = 0; i < alto; i++){
    for(int j = 0; j < ancho; j++){

      if( (mapa[i][j] == SNAPE) || (mapa[i][j] == LUPIN) || (mapa[i][j] ==
MOODY) || (mapa[i][j] == DOLORES_UMBRIDGE) ){
        distancia = i + j;
```

```
        if(distancia < menor_distancia){
          menor_distancia = distancia;

          mas_cerca = mapa[i][j];
        }
        else if(distancia == menor_distancia){
          mas_cerca = VARIOS_CERCA;
        }
      }
    }
  }
  return mas_cerca;
}
```