Parcial: 01-07-2021

Asignatura	Algoritmos y Programación I
Cátedra	Méndez

Ejercicio 1

Las chicas superpoderosas se encargan de mantener la ciudad libre de criminales, es por eso que cada vez que atrapan a uno se encargan de llevarlo a la cárcel. El problema es que se está poniendo cada vez más tedioso organizar las celdas correctamente. Podemos pensar a la cárcel como una matriz de caracteres donde cada posición es una celda y el caracter guardado indica que tipo de criminal hay encerrado.

Sabemos que hay tres tipos de criminales: - Normales: Representados como una **N**. - Super Villanos: Representados como una **V**. - Secuaces: Representados como una **S**.

Para saber si la cárcel está bien organizada hay que asegurarse que ningún secuaz esté encerrado justo al lado de un super-villano (en cualquiera de las cuatro direcciones).

Implementar una función que reciba una matriz de 20×20 y devuelva true si la cárcel está bien organizada o false en caso contrario.

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

#define MAX 20

const char NORMAL = 'N';
const char VILLANO = 'V';
const char SECUAZ = 'S';

/*
 * Precondiciones: los tres parámetros deben estar inicializados.
 * Postcondiciones: devuelve verdadero si en alguna de las posiciones adyacentes de
 * un villano hay un secuaz, de lo contrario devuelve falso.
 */
bool secuaz_cerca(char carcel[MAX][MAX], int fil, int col){
  return (carcel[fil+1][col] == SECUAZ || carcel[fil][col+1] == SECUAZ ||
          carcel[fil-1][col] == SECUAZ || carcel[fil][col-1] == SECUAZ);
}

/*
 * Precondiciones: -
```

```
 * Postcondiciones: devuelve verdadero si ningun villano tiene un secuaz encerrado
 * cerca, de lo contrario devuelve falso.
 */
bool esta_organizada(char carcel[MAX][MAX], int filas, int columnas){

  int i = 0, j = 0;
  bool organizada = true;

  while(organizada && i < filas){
    while(organizada && j < columnas){
      if(carcel[i][j] == VILLANO && secuaz_cerca(carcel, i, j))
        organizada = false;
      j++;
    }
    i++;
  }

  return organizada;
}
```

Ejercicio 2

Las chicas superpoderosas están planeando buscar a las mejores heroínas de todo el mundo, porque saben que sus enemigos se están reuniendo y conspirando contra ellas.

Una de las heroínas que contactaron para ayudarlas, les pidió un listado de todos los enemigos que conocen hasta ahora, por eso los anotaron en un vector y definieron qué rasgos creen ellas que son importantes:

```
typedef struct enemigo {
  char nombre[MAX_NOMBRE];
  bool tiene_superpoderes;
  int veces_derrotado;
  int veces_ganadas;
} enemigo_t;
```

Explicar con tus palabras cómo harías para ordenar el vector de enemigos, ascendentemente por nombre. No está permitido escribir código ni usar ninguna de las palabras reservadas de C.

Se pueden usar alguno de los campos como criterio para ordenar el vector: alfabéticamente, veces derrotado o veces ganado.

Ordenémoslo según *las veces que un enemigo fue derrotado* para que en la primera posición del vector quede el enemigo con menos derrotas, es decir, el más fuerte.

Ahora nos tenemos que preguntar, ¿que ordenamiento usamos? Hemos aprendido: inserción, burbujeo y selección. Usemos *burbujeo*.

Hay que tener en cuenta que tenemos un vector de enemigos, en cada índice del mismo, hay un dato de tipo enemigo (mostrado en el enunciado). Por lo tanto, en cada comparación se debería acceder al campo que lleva la cuenta de las veces que ese enemigo fue derrotado.

Enumeremos los pasos para la primera vuelta:

1. Tomar el primer enemigo del arreglo de enemigos como elemento actual.
2. Comparar las veces que ese enemigo fue derrotado con el segundo enemigo del arreglo.
3. Si el primer enemigo fue derrotado más veces que el segundo, entonces se intercambian de posición. De lo contrario, no intercambiar.
4. Tomar el elemento que sigue como elemento actual.
5. Repetir los pasos 2 y 3 hasta llegar al final del arreglo.
6. Si hubo intercambios, repetir los pasos desde el primer elemento del arreglo.
7. Repetir hasta que se logre una pasada sin intercambios.

Una vez que tenían la lista terminada, Burbuja se acordó que hace poco le ganaron una vez más a Mojojojo y no lo habían contado. Crear un procedimiento que busque a Mojojojo y sume uno, a las veces_derrotado.

```
#include <stdio.h>

#define MAX_ENEMIGOS 100

typedef struct enemigo {
  char nombre[MAX_NOMBRE];
  bool tiene_superpoderes;
  int veces_derrotado;
  int veces_ganadas;
} enemigo_t;

/*
 * Precondiciones: -
 * Postcondiciones: agrega al enemigo cuyo nombre es Mojojojo una derrota.
 */
void sumar_victoria(enemigo_t enemigos[MAX_ENEMIGOS], int tope){

  int i = 0;
  bool sumada = false;
```

```
  while(!sumada && i < tope){
    if(strcmp(enemigos[i].nombre, "Mojojojo") == 0){
      veces_derrotado++;
      sumada = true;
    }
    i++;
  }
}
```

Ejercicio 3

Justo, el humano mala onda de Coraje, está aburridísimo viendo la televisión, en los días cómo hoy, donde nada de lo que ve le gusta, hace lo siguiente:

1. Pone el canal 0, espera a que en la pantalla aparezca un número y va a ese canal.
2. Luego espera que aparezca un número y va a ese canal.
3. Luego espera que en ese canal aparezca un número y va a ese canal.
4. Y así sucesivamente hasta que el número que aparece es el 37.

Crear una función recursiva que reciba un vector de enteros (donde la posición del vector es el canal y el contenido es el número que aparece en pantalla), simule lo que hace Justo y devuelva cuántas veces tuvo que cambiar de canal para llegar al canal que se queda viendo.

```
#include <stdio.h>

#define MAX_CANALES 50
const int NUMERO_BUSCADO = 37;

/*
 * Precondiciones: el canal debe estar inicializado y debe ser > 0.
 * Postcondiciones: devuelve la cantidad de veces que Justo tuvo que cambiar el canal
 * hasta llegar al canal con número 37.
 */
int cambios_canal(int contenido_canales[MAX_CANALES], int canal){

  if(contenido_canales[canal] == NUMERO_BUSCADO) return 0;

  return 1 + cambios_canal(contenido_canales, contenido_canales[canal]);
}
```

Ejercicio 4

El Señor Conejo tiene ganas de buscar otra mansión, pero necesita que la casa no sea de las más antiguas, porque son las que suelen tener más problemas.

Teniendo el código del Señor Conejo para reconocer la casa más antigua, enumerar qué buenas prácticas no se cumplen, y explicar qué error de implementación hay y cómo lo solucionarías.

```
#define CINCUENTA 50

typedef struct casa {
  char direccion[CINCUENTA];
  int anio_construccion;
} casa_t;

/*
 * Precondiciones: -
 * Postcondiciones: devuelve la casa mas antigua del vector de casas en base a su año
 * de construccion.
 */
casa_t casa_mas_antigua (casa_t casas [MAX_CASAS], int t) {

  int anio_mas_antigua = 2000;
  casa_t mas_antigua;

  for (i = 1; i < t; i++) {
    if (casas[i].anio_construccion < anio_mas_antigua) {
      anio_mas_antigua = casas[i].anio_construccion;
      mas_antigua = casas[i];
    }
    i++;
  }
  return mas_antigua;
}
```

1. No se entiende el significado de la constante por su nombre poco descriptivo. En tal caso, se podría llamar `MAX_DIRECCIONES`.
2. Tendría mas sentido que el vector de direcciones sea de tipo `int`.
3. Falta definir la constante `MAX_CASAS`.
4. La variable `int t` tiene nombre no descriptivo respecto a lo que representa. Debería llamarse algo como `tope`, `tope_casas`, `cantidad_casas`, o similar.
5. Dentro del `for()` se hace un `i++`. Eso, en tal caso, podría ser para las estructuras iterativas `while()` o `do...while()`, pero en el caso del ejercicio se esta usando un `for()` y la actualización ya se hace en la declaración del principio.
6. El contador del bucle comienza en 1 y debería comenzar en 0.
7. La lógica del ejercicio no es correcta. El objetivo es buscar la casa con máxima antigüedad de un vector de casas, en otras palabras, ***habría que***

buscar el máximo de un vector.

```
#include <stdio.h>

#define MAX_DIRECCIONES 50
#define MAX_CASAS 100

typedef struct casa {
  int direccion[CINCUENTA];
  int anio_construccion;
} casa_t;

/*
 * Precondiciones:
 * Postcondiciones: devuelve la casa mas antigua del vector de casas.
 */
casa_t casa_mas_antigua(casa_t casas [MAX_CASAS], int tope) {

  casa_t mas_antigua;
  int anio_mas_antigua = casas[0].anio_construccion;

  for(i = 0; i < tope; i++){
    if(casas[i].anio_construccion > anio_mas_antigua) {
      anio_mas_antigua = casas[i].anio_construccion;
      mas_antigua = casas[i];
    }
  }

  return mas_antigua;
}
```

Ejercicio 5

Los sueños son una experiencia humana universal que puede describirse como un estado de conciencia caracterizado por acontecimientos sensoriales, cognitivos y emocionales durante el sueño.

Las pesadillas, son una variante de sueño, donde quien sueña, no la pasa demasiado bien, tal es el caso de Panda, quien en el último tiempo está teniendo demasiadas. Su terapeuta le encomendó que las transcriba, para poder analizarlas. Lo que sí se acuerda, es que todas, tienen un cazador que lo persigue.

Crear un registro `pesadilla_t` para que Panda pueda transcribirlas, incluir todos los campos que considere que una pesadilla puede tener, por ejemplo lugar, estado del tiempo, nivel de sufrimiento, y como estamos hablando de Panda, un cazador. Se ruega ser creativo y no limitarse sólo a la información del enunciado

```
typedef struct pesadilla{
  char lugar[MAX_LETRAS];
  int porcentaje_sufrimiento;
  char cazador;
  char duracion; // B: breve - I: intermedio - L: larga
  char clima; // N: nublado - S: soleado - R: lluvioso
  bool grito;
}pesadilla_t;
```

Dado un vector de pesadillas, determinar qué porcentaje de ellas, son un día nublado y el cazador es Polar.

```
#define MAS_PESADILLAS 100

const char NUBLADO = 'N';
const char POLAR = 'P';

/*
 * Precondiciones: -
 * Postcondiciones: devuelve el porcentaje de pesadillas que cumplen con la
   caracteristica de que en la pesadilla este nublado y el cazador sea polar.
 */
int porcentaje_segun_caracteristicas(pesadilla_t pesadillas[MAX_PESADILLAS], int tope){

  int contador_pesadillas = 0;

  for(int i = 0; i < tope; i++){
    if(pesadilla[i].clima == NUBLADO && pesadilla[i].cazador == POLAR)
      contador_pesadillas++;
  }

  return (contador_pesadillas*100/tope);
}
```