# REPORTE PSETS Y LABS

*Cinthia Carolina Duarte Ruiz.*

*Y21C2-CDUARTE*

## PSETS

**Pset 0**: BlackPink Light Catch

Debes atrapar tantos lightstick de blackpink (negros y rosas) como puedas. ¡Ten cuidado de tocar los de Twice (amarillos y rosas)! Perderás una valiosa vida.

La caída de los lighsticks es controlada por las posiciones x & y del plano. Seleccionarán una posición random a lo largo de todo el juego, así también es asignado una velocidad por nivel. El sprite se moverá solamente de izquierda a derecha. Cada que toque un lightstick rosa y negro se sumará un punto. Si toca uno amarillo y rosa se restará una vida hasta que se acaben. Tiene un total de 4 niveles.

**Pset 1:**

* Hello:

Implementando cs50.h con la función get\_string es pedido su nombre al usuario, seguido de imprimirlo junto con un saludo. “Hello, Cinthia!”

* Mario-more:

Es pedido al usuario la altura de la pirámide validando que se encuentre dentro del rango [1,8]. Se usa el ciclo for para iterar en el rango de la altura ingresado, así mismo serán anidados 3 ciclos for más: El primero para imprimir los espacios, el segundo para imprimir hashes y el tercero para duplicar la pirámide. Por último, es impreso un salto de línea.

* Credit:

Definiendo una variable de tipo long donde se guardará el número de tarjeta del usuario. Es contada la longitud del número con ayuda de un ciclo while.

Si el contador es diferente a 13 y 15 y 16 entonces es inválida.

Luego se utiliza el algoritmo de Luhn para verificar que tipo de tarjeta es. Por último, se imprime la tarjeta y si no pertenece a ninguna de las existentes es inválida.

**Pset 2:**

* Readability:

Si el texto ingresado es parte del alfabeto se cuenta como letra, si no es omitido.

Si hay un espacio dentro de la longitud del texto se cuenta como una palabra más.

Si hay un punto, un signo de interrogación o admiración, es una oración más.

Dependiendo de estos 3 puntos, se hace el cálculo para indicar a que grado pertenece el texto.

* Caesar:

Primero se asegura el uso correcto de la llave. Si los argumentos son diferentes a 2 o si la llave es diferente a un dígito. Luego se convierte la llave a un entero.

El proceso de encriptación comenzará siempre y cuando el texto sea un alfabeto. Si es mayúscula es encriptado en dependencia de su código ASCII, así mismo si es minúscula. De otra manera será impreso tal cual.

**Pset 3:**

* Plurality:

*Vote:* Con un ciclo for puedes iterar sobre la cantidad de candidatos. Si el nombre dado está en la lista de candidatos entonces se le asignará un voto.

*Print\_winner:* Se verifica la mayor cantidad de votos. Si los votos del candidato son igual a la mayor cantidad entonces se imprime el nombre del candidato.

* Runoff:

*Vote:* Si el candidato se encuentra en las opciones entonces será guardado en las preferencias del votante y su rank de favoritos.

*Tabulate:* Recorres la cantidad de votos y de candidatos y verificas si el candidato no está eliminado. Si es así entonces los votos según la preferencia del votante aumentarán.

*Print\_winner:* Si el candidato tiene más de la mitad de votos de preferencia entonces será el ganador.

*find\_min:* Si el candidato no ha sido eliminado entonces se asigna a MAX\_VOTER la cantidad de sus votos.

*is\_tie:* Si el candidato no ha sido eliminado y el número de votos del candidato es distinto al mínimo retorna false.

*eliminate:* Si los votos del candidato son iguales a min entonces está eliminado.

**Pset 4:**

* Filter-less

Grayscale: Usando un ciclo for se recorre la altura y anchura de la imagen. Se calcula el promedio del valor del pixel y se asigna a los colores RGB de la imagen.

Sepia: Usando la fórmula para los colores sepia se compara si el resultado es mayor a 255. Si no es así significa que el pixel no tiene filtro y se asigna el valor de la fórmula.

Reflect: Guarda la imagen original temporariamente en un variable de tipo RGBTRIPLE y luego refleja la imagen derecha a la izquierda invertidamente.

Blur: Crea una imagen temporaria y recorre los pixeles verticalmente y horizontalmente, verifica si son válidos y están dentro de las filas y columnas y almacena el valor de ellos en una variable. Luego se redondean y asignan a la imagen temporaria.

Por último, recorre la original y la reemplaza por la temporaria.

* Recover

Es necesario primero validar la entrada de argumentos, si es diferente a 2 entonces cortará exitosamente el programa. Luego se abre un archivo tipo FILE en modo de lectura.

Definimos una array buffer de tipo unsigned char con valor de 512 bytes, otro archivo tipo FILE que tendrá como valor definido NULL y otro array tipo char.

Dentro de un ciclo while se lee los 512 bytes en el buffer. Si se encuentra el inicio de un nuevo JPEG, se determina que hay una imagen y se cierra el archivo. Luego se imprime el JPEG en el archivo nuevo en forma de 001.pg, 002.jpg y así sucesivamente.

Verifica si en realidad se impreso una imagen y la sobrescribe en el buffer.

Por útimo cierra todos los archivos que previamente fueron abiertos.

**Pset 5:** Speller

hash: Esta función recibe como argumento la palabra que se buscó y mediante y un ciclo for guarda en un arreglo un código hash de la palabra.

check: Recibirá como argumento la palabra del diccionario y verificará si se encuentra en él. Guardamos en una variable entera lo que retorna nuestra función hash y creamos un nodo que contendrá la lista donde se almacena el hash de la palabra.

Mientras que el nodo que definimos sea diferente a NULL, verificamos que la palabra sea igual al nodo que apunta a ella. Luego al nodo se le asignará el valor de él mismo, pero ahora apuntando al siguiente.

Load: Recibe como argumento el diccionario y se abre en un nuevo archivo de tipo FILE en modo de lectura.

Con un ciclo while leemos carácter por carácter el diccionario hasta que se acabe. Declaramos un nuevo nodo que tendrá un espacio de memoria dinámico de un nodo. Luego se copia la palabra en el nuevo nodo. Este mismo apuntará al nodo siguiente y este contendrá el valor hash de la palabra, así recíprocamente el índice donde se encuentra el hash en la lista será asignado el valor del nodo mientras su tamaño se va aumentando. Cerramos todos los archivos que anteriormente abrimos.

Size: Retorna el tamaño del diccionario que en la función load aumentó.

Unload: Recorre el número de espacios en el hashtable. Define un nodo que contendrá el valor de la lista en el índice correspondiente. Mientras que el nodo exista se creará un nodo temporal que guardará el valor del nodo. Luego el nodo original apuntará al siguiente. Por último, liberamos el nodo temporal.

Si el nodo ya no existe entonces el diccionario ya estará descargado de la memoria.

**Pset 6:**

* DNA

Importamos Reader y DictReader de la biblioteca csv para el procesamiento de archivos csv, y argv y exit sys para manejar argumentos de línea de comandos y códigos de salida.

Comenzamos asegurándonos de que se proporciona el número correcto de argumentos de la línea de comando usando len(argv). Si se especifica una cantidad incorrecta, se puede llamar al método exit. Si se aprueba esta verificación, se asignan nombres de variables a la BD y argumentos de secuencia. Abrimos los archivos y los convertimos a formatos en los que se pueda trabajar. Primero el archivo csv con el método DictReader, convierte cada fila del archivo en un diccionario. El archivo txt con una sola fila, se puede leer mediante el método read(). Esto devuelve el contenido del archivo como una cadena.

Definimos una matriz vacía que almacenará la secuencia máxima consecutiva de cada STR Max\_count[]. Recorremos cada uno de los STR y verificarlos en secuencia, pero se debe empezar desde 1 ya que la columna 0 almacena los nombres de las personas. Aquí, el STR que se verifica, se almacena en una variable y se agrega un nuevo valor a la matriz que almacena las longitudes de las secuencias más largas para cada STR.

El bucle interno itera a través de la secuencia que se está evaluando y para cada carácter verifica si sigue el STR. Si es así, se verifican los siguientes caracteres para ver si el STR se repite, agregando uno al conteo por cada repetición. Si el recuento es mayor que el máximo actual, se sobrescribe el máximo actual para ese STR.

Con un valor adicional que se agrega a la matriz de conteos máximos para cada STR en la base de datos, la matriz final se puede comparar con las filas de la base de datos.

Aquí se usa el diccionario, ya que habrá un diccionario para cada persona en la base de datos. Se recorre la lista de STR y la matriz de conteos máximos se compara con las secuencias de STR máximo para esa persona. dict\_list[i] representa un diccionario y [reader.fieldnames[j]] devuelve la clave que se buscará en ese diccionario.

Si hay una coincidencia, se incrementa el número de coincidencias. Si el número de coincidencias es igual al número de STR en la base de datos, la secuencia que se está comprobando coincide con alguien en la base de datos y se puede devolver el nombre de la persona. De lo contrario, se imprime 'Sin coincidencia'.

#### Mario-less

Se solicita la altura de la pirámide con get\_int, si esta es negativa o supera el número máximo, se le volverá a pedir la altura. Se hace un primer ciclo que inicia desde row (0) hasta el numero ingresado por el usuario, y luego dentro de este se crea otro ciclo que toma el valor de la altura menos el valor del ciclo anterior restándole a este para imprimir los espacios vacíos, luego un tercer ciclo que comienza de cero hasta que el valor del primer ciclo sea menor o igual a este imprimiendo los “#” de la pirámide, luego de terminar esa línea hace un salto de línea para repetir el procedimiento.

#### Cash

Ponga el contador de monedas a 0.

Solicitar al usuario una cantidad.Si la entrada no es mayor a 0, se pregunta de nuevo.

Se la cantidad a centavos y comience a eliminar, es decir, por cada 25c en la entrada dada de, digamos, X, el contador de monedas aumenta y repetimos esto para 10c, 5c y 1c. Después imprimimos el número de monedas.

#### Readability

solicitamos al usuario una cadena de texto. Despues Inicializamos los 3 contadores diferentes a 0: contador de letras, palabras y oraciones. Iteramos a través de cada letra individual usando el len() .Para cada iteración, contamos: no. de letras, n. de palabras y no. de oraciones

Calculamos el índice usando 0.0588 \* L - 0.296 \* S - 15.8 y redondeamos

Si el número de índice resultante es 16 o superior, escribimos "Grado 16+"; si el número de índice es menor que 1, "Antes del grado 1". De lo contrario, el propio número de índice.

#### Mario-more

Lo mismo para Mario less, lo único que estamos agregando es un Hash alineado a la derecha.

comenzaremos desde donde paramos la solución en Mario less luego imprima dos espacios de impresión (" ") y asegúrese de que el final esté configurado en "", para anular la impresión de una nueva línea predeterminada luego iterar otro bucle for con una fila para imprimir el hash alineado a la derecha.

**Pset 7:**

* Movies

Se inicia con lo básico de sql, que serían las consultas como por ejemplo las palabras SELECT, FROM y WHERE.

Si la pregunta pide ser más específica, por ejemplo, para ordenar la salida alfabéticamente por título, puede usar el comando "ORDER BY" seguido de ASC o DESC, lo que significa orden ascendente o descendente respectivamente. Otra característica útil es el comando LIMIT que restringe la cantidad de salida.

A menudo, nuestros datos no están bien ordenados en un lugar específico. Aquí es donde entra en juego el comando JOIN.Puedo usar JOIN en tantas tablas como sea necesario, siempre que haya un identificador único común entre las tablas que se están uniendo. Estos identificadores pueden tener diferentes nombres de columna, pero sus valores deben ser los mismos.

Las subconsultas son la herramienta final que necesitará para completar Movies. Una subconsulta, también conocida como consulta anidada, es esencialmente una consulta dentro de una consulta. Se puede agregar a varios comandos, incluidos SELECT, FROM o WHERE, así como otros comandos.

* Fiftyville

Es básicamente un juego de detectives de SQL, donde debemos investigar y recuperar la información necesaria para resolver el misterio.

¿Quién es el lardon?

¿A qué ciudad escapó el ladrón?

Quién es el cómplice del ladrón que los ayudó a escapar

PASO 1 - CONOCER CÓMO ESTÁ ESTRUCTURADA LA BASE DE DATOS

Lo primero que hay que hacer es saber cuántas tablas y qué tipo de tablas tiene la base de datos de Fiftyville. Entonces, en la terminal, nos aseguramos de estar en el conjunto de problemas de Fiftyville e iniciamos SQLite para abrir el archivo de la base de datos usando esta línea de comando: sqlite3 Fiftyville.db Luego podemos usar .schema para verificar el esquema de nuestra base de datos.

De esta forma ahora sabemos que la base de datos Fiftivylle tiene 10 tablas

PASO 2 - CONOZCA MÁS INFORMACIÓN DE CRIME REPORTES

Dado que la única información que tenemos es una fecha y una calle, necesitamos más para avanzar en las investigaciones; el primer lugar para comenzar es la tabla de informes de delitos, que podemos navegar fácilmente con la fecha y el estado de la calle que ya tenemos

PASO 3 - CONOCER MÁS INFORMACIÓN BASADO EN LAS TRANSCRIPCIONES DE LAS ENTREVISTAS DE LOS TESTIGOS

Todavía necesitamos más información, por lo que, utilizando las anteriores, podemos navegar la tabla de entrevistas en la base de datos para conocer las transcripciones de los testigos. También debemos mencionar la palabra clave del juzgado en su columna de transcripción como condición. De esa manera solo obtendremos los 3 testigos que necesitamos verificar

PASO 4 - INVESTIGACIÓN DE LOS REGISTROS DE LOS TESTIGOS

Como tenemos un nuevo conjunto de información, necesitamos organizarlos, así que los busco individualmente para saber con seguridad qué tipo de información recupero de las pistas.

Podemos empezar por el primer testigo, buscando el coche que sale del juzgado mencionado en su expediente haciendo referencia a la fecha y hora del robo

PASO 5 - SABER EL NOMBRE DEL LADRÓN

Este es el paso más complicado, necesitaremos unir muchas tablas y consultas. Lo primero que debe hacer es comenzar con la tabla de personas, ya que es donde se almacena el nombre del ladrón y son los datos que queremos generar.

Luego cruzamos la información obtenida de los 3 testigos: número de placa, número de cuenta bancaria, número de teléfono y número de pasaporte.

PASO 6 - CONOCIENDO LA CIUDAD DONDE EL LADRÓN ESCAPÓ

Básicamente, ya tenemos la respuesta aquícuando buscábamos el vuelo mencionado por el tercer testigo en la llamada del ladrón, encontramos su ID de destino de vuelo. Entonces solo necesitamos averiguar qué ciudad representa la ID en la tabla de aeropuertos

PASO 7 - SABER EL NOMBRE DEL CÓMPLICE

Este paso se puede completar fácilmente después de encontrar el nombre del ladrón, solo necesitamos buscar nuevamente en la tabla phone\_calls agregando a la búsqueda anterior el nombre del ladrón y su número de teléfono.

**Pset 8:**

**Homepage**

Index.html

Presentación de la página con escritura animada a base de CSS.

Me.html

Despliega cartas alineadas con imágenes ilustrativas de mis intereses personales.

Songs.html

Despliega cartas alineadas con algunas canciones que escucho seguido recientemente. Incorpora botones de likes funcionales hechos con javascript, donde se retribuyen los botones y cambia su estado al agregar un evento click.

Places.html

Despliega cartas alineadas con imágenes ilustrativas de algunos platillos de la gastronomía asiática que me encantaría probar.

**Pset 9:**

* register

Para la página de registro, necesito crear un formulario que solicite al usuario que ingrese un nombre de usuario, contraseña y confirmación de contraseña. Los formularios en HTML pueden usar el método POST o el método GET. La mayoría de los formularios que he creado utilizan el método POST. Con POST, los datos del formulario aparecen dentro del cuerpo del mensaje de la solicitud HTTP.

* quote

Esta página permite al usuario buscar el precio actual de una acción. El quote.html requiere que el usuario ingrese el símbolo de una acción. Una vez que el usuario envió un símbolo, verificaremos si el símbolo de acciones realmente existe. Si no existe, devolveremos un mensaje de error. Si es así, utilizaremos la función de búsqueda dentro del archivo helpers.py para buscar y recuperar información sobre las acciones. A continuación, mostraremos la información en quoted.html.

* buy

El buy.html requiere que el usuario ingrese el símbolo de una acción y la cantidad de acciones que el usuario desea comprar. El buy.html tiene el mismo formato que el quote.html, excepto que tiene dos campos para que el usuario ingrese datos en lugar de uno. La función buy() dentro de app.py verifica si el símbolo existe y se asegura de que el número de acciones proporcionadas sea un número entero positivo.

* index

El index.html es la página de inicio. Muestra una tabla HTML que resume las acciones que posee el usuario, el número de acciones que posee, el precio actual de cada acción y el valor total de cada participación. También muestra el saldo de efectivo actual del usuario junto con un total general (valor de las acciones + efectivo).

* sell

sell.html es exactamente igual que buy.html. La función sell() comprueba inicialmente si el usuario tiene suficientes acciones para vender. Luego actualiza el historial, el efectivo del usuario y la cartera del usuario y devuelve al usuario a la página de inicio.

* history

history.html es similar a index.html donde muestra una tabla HTML que resume todas las transacciones del usuario. Cada fila muestra el símbolo de la acción, el precio de compra/venta, la cantidad de acciones compradas/vendidas y la fecha y hora en que ocurrió la transacción. La función history() simplemente selecciona toda la tabla de historial y la pasa a history.html.

* (toque personal)

Por último, el conjunto de problemas también requiere que implemente una característica adicional a gusto del estudiante

## LABS

**Lab1:** Population

Pide al usuario el tamaño inicial y final de la población. Mientras el tamaño inicial sea menor al final aplicamos los cálculos correspondientes, así mismo los años aumentarán.

Por último, imprime el número de años que tomará.

**Lab2:** Scrabble

Compute\_score: Recorre la palabra y verifica si es mayúscula. Si esto es cierto, busca la letra en el arreglo de capitales con su código ASCII. Asigna puntos en un arreglo temporario y lo suma al puntaje. Si es minúscula hace lo mismo, esta vez con el arreglo de letras pequeñas.

Si no es alfabeto lo ignora.

Main: Si el puntaje del jugador 1 es mayor al del jugador 2 lo imprime como ganador y viceversa. Si no es así entonces es un empate.

**Lab3:** Sort

Ejecutando la palabra time antes del tipo de ordenamiento y los números podemos medir el tiempo de cada uno. Siendo así, el primer ordenamiento es de burbuja (Bubble Sort) ya que es el que más tiempo se tomó. El segundo es de mezcla (Merge Sort) ya que es el más rápido de los tres. El tercero es selección (Selection Sort), no es tan rápido como el de mezcla, pero sin duda más que el de burbuja.

**Lab4:** Volume

Copiamos el header del archivo de la entrada al de salida con el tipo uint8\_t y las funciones fread y fwrite. Definimos un buffer de tipo int16\_t y lo leemos con ayuda de un ciclo while y la función fread. Aplicamos el factor que contendrá la entrada de argumentos que proporcionó el usuario al buffer y lo sobrescribimos en el archivo de salida.

Por último, cerramos los archivos.

**Lab5:** Inheritance

create\_family: Alocamos memoria dinámicamente para una nueva persona de tipo person. Si hay más de una generación entonces la nueva persona apuntará a los padres en determinada posición. A este se le asignará el resultado recursivo de la función create\_family.

Asigna alelos al azar a la nueva persona basado en los alelos de los padres. Si no hay ptra generación entonces establecemos los punteros de los padres a NULL y asignamos alelos al azar a los hijos que hayan.

free\_family: usamos recursividad para liberar los padres de la persona y a la persona con la función free.

**Lab6:** World cup

#### Simulate\_tournaments

Utiliza un ciclo while que condiciona si hay más de un equipo, dentro de él guarda en una variable el retorno de los ganadores y luego retorna el primer equipo.

#### Main

Recorremos n cantidad de torneos y guardamos en una variable el retorno de la función simulate\_tournaments. Si lo que contiene esta variable se encuentra en los conteos entonces se sumará uno, si no es así entonces el conteo del equipo será solo uno.

**Lab7:** Canciones

1. Realiza una consulta a la base de datos y selecciona el nombre de la cnsion

2 Realiza una consulta a la base de datos y selecciona el nombre de la canción ordenado por su tempo

3 Selecciona los nombres ordenados por su duración con un límite de 5

4 Selecciona los nombres que cumplan con los atributos danceability, energy y valence mayor a 0.75

5 Selecciona energy de songs calculando su promedio

6 Selecciona las canciones donde el artista sea post malone

7 Selecciona energy de songs calculando su promedio siempre y cuando el artista sea Drake

8 Selecciona las canciones que tengan colaboraciones

**Lab8:** Trivia

La función answerHTML -> si el id del botón es "correcto" entonces va a imprimir en una etiqueta la frase Correcto, de lo contrario imprime incorrecto.

Función reserButtonsState -> selecciona todos los botones y elimina las clases correcto e incorrecto

Función getAnswerButton -> Llama a la función resetButtonsState.

Si el id del evento es "correcto" entonces añade la clase "correcto" al botón y llama a la función answerHTML. De lo contrario hace lo mismo, pero con la clase "incorrecto"

DOMContentLoaded -> itera sobre los botones y añade un evento click. Si el resultado del input es igual a switzerland entonces llama a la función answerHTML.

**Lab9:** Cumpleaños

Si el método es Post retribuye el nombre mes y día de los forms.

Si no son llenados entonces se manda un mensaje. De lo contrario se inserta en la base de datos, se seleccionan los cumpleaños y renderiza la plantilla HTML. Si no es así solo selecciona los cumpleaños de la tabla y rendiría la misma plantilla retornando el resultado de la consulta al template