Enunciado:

Resuelva los siguientes ejercicios en C++14 sobre aplicaciones de las estructuras de datos: `vector`, `stack` y `queue`. Los ejercicios más complejos son los que tienen número par.

1. Un histograma muestra la frecuencia con la que ocurren valores en una muestra de datos. Escriba un programa que lea un archivo de texto y cree un histograma con la frecuencia de la aparición de las vocales. Su implementación debe poder manejar tanto letras mayúsculas como minúsculas e incluirlas en el histograma. Por ejemplo, un posible resultado de su programa sería:

```
a: 25
e: 62
i: 47
o: 23
u: 2
```

   que muestra que en el archivo la letra `a` aparece 25 veces, la letra `e` 62 ocurre veces, etc. Utilice objetos de la clase `vector` para almacenar el histograma.

2. Como parte de su interés en series y películas, usted ha decidido registrar la información de varios personajes en un archivo de texto. Cada línea en este archivo tiene un registro que corresponde a un personaje. En cada registro usted mantiene los siguientes campos: serie, nombre, apellido. Estos campos son cadenas de caracteres separadas por comas. Escriba un programa en C++ que permita leer este archivo, almacenar su información en una estructura de datos, e imprimirla en pantalla. Su programa debe:

   a) Tener una función que lea cada línea y retorne el contenido de la misma en un objeto de la clase `vector<string>`, donde cada elemento corresponda a cada uno de los campos en el archivo de texto. El prototipo de su función debe ser:

```
void leerLinea(vector<string> &reg, string linea);
```

   b) La función principal de su programa utiliza la función del numeral anterior para almacenar toda la información del archivo en un objeto de la clase `vector<vector<string>>`, donde cada elemento corresponde a un registro (línea) del archivo.

   c) Tener una función que toma la estructura de datos del numeral anterior e imprime en pantalla su contenido, tal que cada registro se imprime en una línea de texto, y en cada línea se muestran los valores de los campos separados por tabuladores. El prototipo de su función debe ser:

```
void imprimirLineas(vector<vector<string>> regs);
```

   El contenido de un archivo de muestra para este ejercicio se muestra a continuación.

```
Batman, Bruno, Diaz
Star Wars, Luke, Skywalker
```

```
Star Wars, Leia, Organa
Star Wars, Jango, Fett
Harry Potter, Harry, Potter
Harry Potter, Hermione, Granger
Harry Potter, Tom, Riddle
Harry Potter, Ron, Weasley
```

3. Escriba un programa, usando la estructura de datos `stack`, que lee un `string` con caracteres que incluyen paréntesis, llaves y corchetes de `cin`, y determine si estos símbolos están debidamente balanceados. Por ejemplo, su programa, implementado como un predicado, debe retornar `true` para la cadena de caracteres [()]{}{[()()]()}, pero debe retornar `false` para el `string` [(]).

4. Un punto de empaque funciona de la siguiente manera. En cada momento del tiempo (digamos cada minuto) llega un elemento de tres posibles: agendas, cuadernos y libros. El tipo de elemento que llega es aleatorio con la misma probabilidad para los tres tipos. Una persona mantiene 3 pilas con los elementos que van llegando, una para las agendas, una para los cuadernos y una para los libros. Cada minuto, después de que llega un elemento, otra persona revisa las 3 pilas; si hay al menos 5 elementos en cada pila, esta persona arma un paquete. Cada paquete está compuesto de 3 agendas, 3 cuadernos y 3 libros. La persona arma un paquete retirando los elementos de las pilas y ubica el paquete armado en otra pila de paquetes. En esta pila se pueden acumular hasta 12 paquetes. Cuando se tengan 12 paquetes, se arma una caja y se libera la pila.

   a) Escriba un programa en C++ que implemente el funcionamiento descrito anteriormente. Utilice objetos de la clase `stack` para representar las pilas y asígnele un código (un entero por ejemplo) a cada elemento y cada paquete para identificarlos.

   b) Su programa debe ejecutarse por un número determinado de minutos, identificado como una constante `MINUTOS_SIMULACION`.

   c) Al terminar su programa debe reportar cuántos elementos de cada tipo entraron al sistemas, y cuántos paquete y cajas se despacharon.

5. El problema de Flavio Josefo (*Josephus problem*). Hay n personas de pie en un círculo a la espera de ser ejecutadas. La cuenta comienza en una posición m del círculo y se van ejecutando personas saltando m personas entre medio. El procedimiento se repite con las personas restantes, a partir de la siguiente persona, que va en la misma dirección y omitiendo el mismo número de personas, hasta que sólo una persona, el ganador, permanece (¡Flavio Josefo!). El problema, dado el número de personas n y el punto de partida m, es elegir la posición en el círculo inicial para evitar la ejecución. Escriba un programa que realiza la simulación de este problema usando la estructura de datos adecuada.

6. Un búfer de teclado (*keyboard buffer*) es una sección de la memoria del computador que almacena las pulsaciones de las teclas antes de ser procesadas. Los búferes

de teclado se han usado durante mucho tiempo en el procesamiento de línea de comandos. Cuando un usuario ingresa un comando, lo ve reflejado en su terminal y puede editarlo antes de que sea procesado. Una primera versión de un búfer de teclado puede ser implementada de acuerdo a los siguientes pasos.

a) Lea del teclado, caracter por caracter, un mensaje (comando) ingresado por un usuario.

b) La presencia del caracter `"*"` distinguirá distintos mensajes (comandos).

c) Distintos mensajes (comandos) ingresados por el usuario deben ser imprimidos en diferentes líneas.

d) Una vez el usuario presiona `<enter>` su programa debe imprimir los mensajes (comandos) y terminar.

e) Los pasos relacionados con la lectura de los mensajes (comandos) deben ser implementados como una función.

f) A su vez, los pasos relacionados con la escritura de los mismos requieren ser incluidos en una segunda función.

Como ejemplo, considere el mensaje `"Hola, mundo!*"<enter>` ingresado por el usuario. La salida de su implementación debe tener la forma

```
Hola, mundo!
```

Ahora considere como segundo ejemplo el comando `"Hola*mundo*"<enter>`, en este caso la salida debe ser

```
Hola
mundo
```

Por tanto, si combina los dos comandos anteriores en el mensaje `"Hola*mundo*Hola, mundo!*"<enter>`, la salida final del programa debe ser

```
Hola
mundo
Hola, mundo!
```

Usando la estructura de datos idónea, implemente una primera versión de un búfer de teclado que tenga la funcionalidad anteriormente descrita y demuestre su correcto funcionamiento. El funcionamiento de su implementación debe ser tal que las letras aparezcan en la pantalla en el orden que fueron ingresadas por el usuario desde el teclado, y reproduce los ejemplos anteriores. Para guiarse en su implementación, haga uso del archivo que se adjunta con este ejercicio. Por último, tenga en cuenta la siguiente información sobre la implementacion adjunta con este ejercicio:

a) Los pasos relacionados con la lectura de los mensajes (comandos) deben ser implementados como una función `store` que lea de la entrada estándar y almacene los caracteres en una estructura de datos.

b) Los pasos relacionados con la escritura de los mensajes (comandos) deben ser implementados como una función `retrieve` que extraiga los caracteres de la estructura de datos y los imprima en pantalla, cada comando en una línea diferente.

7. Cada comando en Linux devuelve un estado de salida o *exit code* (a veces denominado estado de retorno o código de salida). Un comando exitoso devuelve un `0`, mientras que uno no exitoso devuelve un valor distinto de cero que generalmente se puede interpretar como un código de error. Los comandos, programas y utilidades de Linux que se comportan bien devuelven un código de salida `0` al finalizar con éxito, aunque hay algunas excepciones. Una lista de errores típicos se encuentra en este enlace. Estas listas son típicamente implementadas como un `map`. Escriba un programa que imprime los códigos de retorno junto con el mensaje asociado, mostrados en la lista del enlace. Para esto

   a) Genere un archivo de texto con esta lista, el cual debe ser leído por el programa.

   b) Guarde el contenido del archivo en la estructura de datos correspondiente.

   c) Una vez definida la estructura de datos, imprima su contenido.

   Su programa debería tener una salida como la mostrada a continuación:

```
Exit code       Message
1               Catchall for general errors
...             ...
255*            Exit status out of range
```

8. La criba de Eratóstenes (*sieve of Eratosthenes*) es un algoritmo que permite encontrar todos los primos menores que un número natural $N \in \mathbb{N}$. Usando este algoritmo, cree una lista que contiene todos los números en el intervalo $[1, N]$, junto con un rótulo que identifica al número como primo o no. Demuestre el correcto funcionamiento de su implementación imprimiendo dos listas: una con los números primos menores o iguales a N y otra con los no primos menores o iguales a N. Por ejemplo, si $N = 10$, la salida de su programa debería ser

```
Prime numbers less than or equal to 10:
2
3
5
7
Non-prime numbers less than or equal to 10:
1
4
6
8
9
10
```

9. Escriba una función

```
std::set<int> createPrimeSet(int max);
```

que retorna un conjunto de números primos entre 2 y `max`. Un número N es primo si tiene exactamente dos divisores, que son siempre 1 y el número N. Para verificar si un número es primo no requiere revisar todos los divisores posibles. Los únicos números que debe verificar son los números primos entre 2 y la raíz cuadrada de N. Use el método que prefiera para encontrar números primos; la criba de Eratostenes es una opción.

10. Dados dos contenedores asociativos (*associative containers*) tipo `std::set`, implemente una función predicado `subset(...a, ...b)` que toma como parámetros dos referencias constantes a conjuntos `a, b` tipo `std::set<...>`, y retorna `true` si el `std::set<...> a` es un subconjunto del `std::set<...> b`. En caso contrario el predicado debe retornar `false`.