Cola de eventos

Muchos programas de usuario deben reaccionar ante situaciones cuya ocurrencia no se conoce de antemano. Por ejemplo: que una persona acerque su teléfono celular al oído, que un mensaje llegue por la red, o que el usuario presione el botón secundario del ratón. A estas situaciones se les llama formalmente **eventos**.

Los programas no pueden suponer un orden en que los eventos ocurren, y por tanto, deben implementar mecanismos que permitan reaccionar apropiadamente en respuesta a estas situaciones. Estos mecanismos se componen de cuatro conceptos provenientes del *paradigma de programación orientado a eventos*: el evento, la cola de eventos, el ciclo de eventos, y el manejador de eventos.

Por ejemplo, cuando el usuario está frente a un editor de texto y presiona una letra en el teclado, el editor querrá reaccionar a este evento y agregar el carácter correspondiente al archivo de texto. Sin embargo, el editor podría estar ocupado realizando otras tareas (por ejemplo, guardando un archivo grande en disco) mientras el usuario presiona la tecla. Para evitar perder esta acción del usuario, el programa (con ayuda del sistema operativo) almacena el *evento* en una **cola de eventos**. Así el evento no se atiende de inmediato en el momento en que ocurre, sino que simplemente se agrega a la cola.

Mientras el programa está ocupado podrían acumularse muchos eventos en su cola de eventos. Cuando el programa finalmente esté libre, le interesará atender los más urgentes primero. Por tanto la cola puede distinguir cuáles eventos son más prioritarios que otros. Por ejemplo, un aviso de batería baja es más prioritario que muchas teclas que se hayan presionado. De esta forma, el programa obtiene el próximo evento más prioritario que esté en la cola y lo atiende.

Atender un evento significa que el programador quiere proveer código que se ejecute cada vez que ese evento se produce. Este código se escribe en una subrutina (método o función) que va a ser invocado no por el programador, sino por la ocurrencia fortuita del evento (como presionar la tecla). Por eso, esta subrutina se le llama **manejador de evento** (*event handler*) por ser invocada como respuesta a un evento. Por ejemplo, en el manejador del evento el programador determinaría la tecla que el usuario presionó y la agregaría al archivo de texto en la posición del cursor.

Como en la *cola de eventos* pueden haber varios eventos esperando, estos son atendidos en un **ciclo de eventos**. Por cada evento se invocan los *manejadores de eventos* que el programador haya asociado. Si la cola se queda vacía, el programa estará ocioso y podrá descansar, lo que ahorra recursos de cómputo. Si al invocar un *manejador de evento* el programa consume mucho tiempo, los nuevos eventos que se generen en el entretanto no se perderán, sino que serán agregados a la cola de espera.

[50%] Cola de prioridad

Implemente una estructura de datos genérica llamada `ColaPrioridad` o en inglés, `PriorityQueue` (este enunciado usa identificadores en inglés, pero puede programar en español, pero por favor no mezcle idiomas). Este contenedor debe almacenar elementos de cualquier tipo de datos clasificados por prioridad. Las

prioridades se representan con valores enteros sin signo, donde un valor menor representa más prioridad que uno mayor, y por tanto, `0` indica la máxima prioridad.

En una **cola de prioridad** un elemento con mayor prioridad siempre es servido antes que un elemento con menor prioridad. Pero si dos elementos tienen la misma prioridad, serán atendidos en el orden en que ingresaron a la cola de prioridad. Usted debe implementar una estructura de datos propia usando el lenguaje de programación C++, y no puede usar los contenedores disponibles en la biblioteca estándar. Su estructura de datos debe:

1. [10%] Almacenar elementos de cualquier tipo de datos. Hay varias formas de implementar una cola de prioridad, como arreglos de colas (recomendado), árboles binarios de búsqueda, y mapas (arreglos asociativos). Cualquiera que escoja, su implementación debe ser *genérica* y eficiente para proveer las siguientes operaciones.
2. [5%] **Constructor**. Permite crear una cola de prioridad. Debe recibir la cantidad de prioridades permitidas por parámetro. Por ejemplo, si se recibe `10`, indicará que se necesitan 10 niveles de prioridad, desde la prioridad máxima (`0`) hasta la menor prioridad (`9`). Implemente el método `count()` que indica la cantidad de elementos alojados en la cola, y el método `empty()` que retorna verdadero si no hay elementos en la cola.
3. [10%] **Agregar a la cola**. Implemente el método `enqueue()` que recibe dos parámetros: un elemento y su prioridad. El elemento debe ser agregado a la cola con su respectiva prioridad en tiempo deseablemente constante, es decir, en el peor caso $O(1)$.
4. [10%] **Sacar de la cola**. Implemente el método `dequeue()` que no recibe parámetros pero retorna el elemento con mayor prioridad almacenado en la cola. Si varios elementos tienen esa prioridad, se retorna el que ingresó primero a la cola. Después de invocar a este método, el elemento retornado es removido de la cola. Debe lanzar una excepción en caso de que se invoque este método cuando la cola está vacía. Este método también debe ser eficiente y NO puede ser $O(N)$ o peor, donde N es la cantidad de elementos almacenados en la cola de prioridad.
5. [15%] **Uso adecuado de memoria**. Su cola de prioridad debe administrar correctamente la memoria (sin producir fugas ni accesos inválidos), y poderse *copiar* en las formas que permite el lenguaje de programación.

[50%] Cola de eventos

Use su cola de prioridad para implementar una *cola de eventos*. La cantidad de eventos a la que un programa puede reaccionar es indefinida, y puede incrementar en el tiempo. Por tanto, su solución debe estar diseñada para poder agregar nuevos tipos de eventos con el menor impacto posible en el código fuente. Para propósitos de esta evaluación, su programa debe permitir dos tipos de eventos: eventos de teclado y eventos de ratón.

6. [15%] **Eventos**. Suponga que todos los eventos conocen la fecha y hora en que fueron generados, la cual se representa con un número entero en formato `YYMMDDhhmmss`, donde `YY` representa los dos últimos dígitos del año, `MM` dos dígitos del mes (01-12), `DD` el día (01-31), `hh` la hora (00-23), `mm` el minuto (00-59), y `ss` el segundo (00-59). Los eventos deben cargarse de un archivo y deben imprimirse en un formato reducido que el usuario solicita, y la cual es distinta para cada tipo de evento. Para el usuario la existencia de un "evento puro" no tiene sentido, sino que tiene que ser de alguno de los dos siguientes tipos.

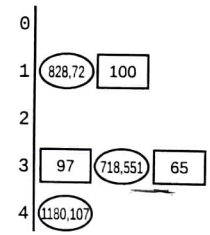
7. [10%] **Eventos de teclado**. Un evento de teclado conoce el código ASCII de la tecla que se presionó. Cuando se imprimen, estos eventos escriben el carácter ASCII correspondiente (y no su código decimal) entre corchetes, de la forma `[X]` donde `X` corresponde al carácter de la tecla presionada.
8. [10%] **Eventos de ratón**. Un evento de ratón conoce las coordenadas (x, y) de la pantalla donde ocurrió el evento de ratón (por ejemplo, un clic). Cuando se imprimen, el usuario quiere que estos eventos lo hagan en formato `(x,y)` donde `x` y `y` corresponden a las coordenadas enteras.
9. [15%] **Programa de pruebas**. Provee un programa de pruebas que lee los eventos de la entrada estándar (como los que se muestran en el ejemplo de entrada) y los almacena en la cola de prioridad que implementó en la primera parte de esta evaluación. Finalmente imprime los eventos en el orden de prioridad como se detalla más adelante.
10. [+10%] **Extra**. Resuelve el problema planteado en el juez en línea (HackerRank). Recuerde tomar fotografías de su solución en papel y transcribiarla al juez en línea.

La entrada consta en la primera línea de la cantidad de niveles de prioridad que el usuario quiere definir. Después de una línea en blanco se proveen los eventos ocurridos, uno por línea. Cada línea tiene una palabra que indica el tipo de evento (`keyboard` o `mouse`), un entero que indica su prioridad, y el momento en que ocurrió el evento en el formato de fecha y hora descrito. Después de esta fecha, vienen los campos específicos de acuerdo al tipo de evento. La figura muestra una representación visual de dichos eventos de acuerdo a sus prioridades usando rectángulos para los eventos de teclado y elipses para los de ratón.

Ejemplo de entrada:

```
5

keyboard 3 191211090006 97
mouse 1 191211090533 828 72
mouse 3 191211090557 718 551
keyboard 3 191211090629 65
mouse 4 191211091011 1180 107
keyboard 1 191211091011 100
```



Ejemplo de salida:

```
(828,72)[d][a](718,551)[A](1180,107)
```

Para corroborar el sistema de colas, el programa de pruebas debe imprimir los eventos en el formato solicitado por el usuario en el orden en que deben ser atendidos de acuerdo a su prioridad.

En todos los rubros se evaluará: las buenas prácticas de programación; uso de identificadores significativos; reutilización de código; eficiencia; uso de copias, punteros, y referencias; evitar conversiones implícitas inadecuadas; indentación; paréntesis y llaves balanceados; y uso adecuado de la palabra reservada `const`.