1. Generar una clase MsgQueue que permita administrar el recurso IPC MsgQueue. Dicha clase deberá poseer métodos para crear el recurso,y para enviar y recibir cadenas de caracteres. Para ello se deberá:
    a. Poseer los miembros privados:
        - Estructura para definir las variables para enviar/recibir.
        ```
        struct mymsgbuf{
              long mtype;
              char mtext[MAX];
        };
        ```
        - Variable para almacenar la llave de acceso al recurso
        `key_t llave;`
        - Variable para almacenar el id de conexión al recurso
        `int id;`
        - Variable indicadora de proceso que borra el recurso
        `bool borrar;`

    b. Crear un constructor parametrizado y por defecto que reciba la información necesaria para obtener la llave de conexión. Este constructor debe crear la llave, conectarse y crear el recurso en caso de no existir. Su prototipo será:
    `MsgQueue(char * p = NULL , char a = 0 , bool destroy = false);`

    c. Crear un método para conectarse a la msgQueue en caso que no se haya conectado en el constructor parametrizado. Su prototipo será:
    `bool  conectar( char * ,char );`

    d. Crear los métodos para enviar y recibir datos por la cola de mensajes:
    `bool  enviarMsj  ( char * , int , long typ = 1) const;`
    `int   recibirMsj ( char * , int , long typ = 0) const;`

    e. Crear el método set para la variable booleana borrar, que indique al proceso si debe o no remover el IPC en su destructor.

    f. Crear el destructor que elimine el recurso en caso que la variable **`borrar`** esté setada.

2. Generar un programa que instancie un objeto de tipo MsgQueue tal como fue descripto en el punto anterior. El programa deberá además preguntar al usuario si desea leer o escribir de la cola de mensajes, y en función de lo que elija el usuario invocar a los métodos correspondientes del objeto. Una tercera opción le permitirá al usuario salir del programa, y el mismo podrá elegir si desea o no borrar el IPC.

    a. Ejecutar 2 veces el mismo programa y verificar que ambos procesos se conecten al mismo IPC. Para esto se puede imprimir el ID de cada MsgQueue (se puede implementar un método getID, en la clase MsgQueue que permita leer este valor).

Asimismo, abrir una tercera terminal y ejecutar el comando ipcs, que permite listar los IPCs abiertos y verificar que el MsgQueue de ID indicado se encuentre en la lista.

b. Con al menos 2 procesos abiertos, enviar y recibir mensajes entre ellos. Generar un tercer proceso y enviar mensajes con diferente `mtype` para verificar su funcionamiento.

c. Finalizar la ejecución de los procesos y verificar que el IPC se haya liberado correctamente mediante el llamado al comando ipcs.

3. Generar una clase IPC que posea la variable llave, la función para obtener la misma, y la variable booleana borrar, también con sus métodos set y get asociados. Modificar luego la clase MsgQueue para que herede de esta clase IPC, eliminando de la misma los métodos que ya se encuentren desarrollados en la clase IPC.

4. Generar una nueva clase `ShMem`, que herede de la clase IPC. La misma deberá poseer:

   a. Variables privadas:
      - Un puntero que indicará el comienzo del bloque de memoria compartida una vez creada y attacheada, y una variable que indique el tamaño del bloque:
        ```
        char * buffer;
        int size;
        ```
      - Variable para almacenar el id de conexión al recurso
        ```
        int id;
        ```

   b. Crear un constructor parametrizado y por defecto que reciba la información necesaria para obtener la llave de conexión. Este constructor debe crear la llave, conectarse y crear el recurso en caso de no existir. Su prototipo será:

```
ShMem(char * p = NULL , char a = 0 , int tam = 0, bool destroy = false);
```

   c. Crear un método para conectarse a la ShMem en caso que no se haya conectado en el constructor parametrizado. Su prototipo será:

```
bool  conectar( int tam , char * ,char );
```

   d. Crear los métodos para escribir y leer de la memoria compartida:

```
bool escribir ( void * msg, int longitud, int inicio typ = 0) const;
    int leer ( char * buf, int longitud, int inicio = 0) const;
```

   e. Crear el destructor que elimine el recurso en caso que la variable **`borrar`** esté setada.

5. Modificar el programa del punto 2 para que en lugar de enviar mensajes mediante un MsgQueue lo haga mediante un ShMem. Modificar luego su comportamiento para que en lugar de enviar cadenas de caracteres envíe estructuras de datos, del siguiente tipo:

```
struct sensores {
        int nro_sensor;
        float medicion;
        };
```

   El proceso escritor deberá colocar en la posición 0 de la memoria compartida un entero con la cantidad de sensores que escribe y a continuación un bloque como el indicado con la medición de cada sensor (una estructura por cada sensor, en posiciones consecutivas de la memoria).

   El proceso lector deberá leer de la memoria compartida las N mediciones e imprimirlas por pantalla.

6. Agregar a la clase ShMem la posibilidad de utilizar semáforos para la lectura y escritura de la memoria. Para ello se deberán agregar las siguientes características:

   a. Variables privadas:
      - Variable para almacenar el id del semáforo
        `int sem_id;`

   b. Un método privado que permita bloquear la memoria compartida, y uno que la desbloquee. El método que intenta bloquear el acceso a la memoria debe quedarse esperando en caso de que haya otro proceso bloqueando la misma, hasta que este la libere:
      `void bloquearShMem();`
      `void desbloquearShMem();`

   c. Modificar los métodos escribir, leer, conectar, y los constructores y destructores para que agreguen la operatoria de los semáforos. En los constructores y el método conectar se deberá generar también el IPC semáforo asociado, con el mismo key que se utiliza para generar la shared memory, y en los métodos de escritura y lectura se deberá invocar a los métodos bloquear y desbloquear antes de hacer la operación de lectura y escritura de la memoria.

7. Sobrecargar los operadores << y >> para poder enviar y recibir distintos tipos de datos a la memoria compartida. Para esto, se deberá contar además con una variable privada en la clase que indique la última posición leída o escrita, y un método para volver este valor a 0 (reset) y de esta manera ir concatenando información. Se busca poder ejecutar el siguiente código:

   - En el proceso escritor:

```
//1 - Instancio un objeto de tipo Shared Memory con los datos de conexión para mi IPC:
ShMem memoria ( … )
//2 - Genero un vector de sensores
sensores medicion[N];

//3 - Cargo los valores de las mediciones en el array de sensores
…

//4 - Guardo en la memoria compartida los valores leídos:
//Primero indico que voy a escribir desde la primer posición:
memoria.reset();
//Luego guardo la cantidad de sensores:
memoria << N;
//Y por último la información de cada uno:
for ( int i = 0 ; i < N ; i ++)
      memoria << medicion[i];

//De acuerdo al sistema se podrían repetir los puntos 3 y 4 en forma cíclica para ir leyendo y enviando todos los sensores periódicamente.
```

- En el proceso lector

```
//1 - Instancio un objeto de tipo Shared Memory con los datos de conexión para mi IPC:
ShMem memoria ( … )

//2 - Genero una variable para saber cuantos sensores tengo y otra para armar un array de sensores:
int cant_sensores;
sensores *mediciones;

//3. Leo la memoria para saber cuantos sensores tengo:
memoria.reset();
memoria >> cant_sensores;

//4. Genero un array dinámico para almacenar la información de los sensores:
mediciones = new sensores[cant_sensores];

//5. Leo la información de los sensores:
for ( int i = 0 ; i < cant_sensores ; i++ )
      memoria >> mediciones[i];
```

```
//6. Imprimo la información de los sensores en pantalla:
…

//De acuerdo a cómo se desarrolle el sistema se podrían repetir los
puntos 3 a 6 en forma cíclica para mostrar en tiempo real el valor
de todas las mediciones.
```