INFORMATICA II	SUBTEMA	TEMA #	PARCIAL	26/06/2021
	C++/IPC	11	# 1	

Estimado alumno,

A partir de la siguiente página encontrará el enunciado de su examen. Le recordamos que ya no puede cambiarlo.

Ud. deberá resolver el ejercicio en su computadora. Luego, deberá subir los archivos realizados comprimidos (preferentemente zip). Si lo desea, puede agregar un documento de texto con explicaciones que quiera hacerle llegar a su profesor.

Cada aproximadamente una hora debe actualizar su repositorio GIT con lo que tenga hasta ese momento resuelto.

Le recordamos:

	Fecha	Tópico	Examen disponible durante	Tiempo para hacerlo desde bajada	Modalidad	Entregables
1	**Sáb 26/6 desde las 8am hasta Dom27 – 19hs.**	C++/IPCs	35hs.	4hs. (*)	Ejercicio en compilador (C++)	Archivo .zip (ver NOTA 1)
3	**Sáb 17/7 desde las 8am hasta Dom18 – 19hs.**	MdE	35hs.	4hs. (*)	Ejercicio en uModel Factory	Archivo .zip (ver NOTA 2)

(*) La duración nominal del examen se cuenta desde el momento en que el alumno comienza a realizar el examen dentro del Aula Virtual (AV). La cuenta del tiempo la lleva la plataforma Moodle automáticamente. Si bien es cierto habrá 35hs a partir del horario de comienzo para comenzar la evaluación, y que las evaluaciones en sí pueden realizarse en 4 horas, debe tenerse presente que, a las 18:59:59 del día domingo quedará bloqueada la subida del archivo final. En criollo: Si va a tomar el examen de MdE a las 18hs del domingo 18 de julio, solo tendrá una hora para entregarlo.

NOTA1: con los fuentes del proyecto/ejercicio + archivo de texto con aclaraciones, de considerarlo necesario el alumno.

NOTA2: con los fuentes del proyecto + el archivo .umf correspondiente a la MdE realizada en uModel Factory + archivo de texto con aclaraciones, de considerarlo necesario el alumno.

Cátedra de Info2

C++:

Dada la siguiente clase PRODUCTO (inmodificable)

```
class PRODUCTO
{
    private:
       char *  Nombre;
       int     Codigo;

    public:
       PRODUCTO( );
       ~ PRODUCTO( );
       void Set_Nombre(char *);
       void Set_Codigo (int);
       char* Get_Nombre(void);
       long  Get_Codigo (void);
       ostream& operator << (ostream &o, PRODUCTO &);
};
```

1. Desarrolle los métodos de la clase “PRODUCTO”
   a. ***void Set_Nombre (char *);*** haciendo uso del concepto de **memoria dinámica**. Si no se pasa ningún parámetro deberá ponerle el nombre “SIN NOMBRE” al producto por default.
   b. ***void Set_Codigo (int);*** Debe chequear que el código ingresado es mayor a 0 y menor a 50000. De no ser asi se asignará el número 0 como código del producto.
   c. ***ostream& operator << (ostream &o,*** PRODUCTO ***&);*** Este método tiene que imprimir la siguiente cadena “PRODUCTO NOMBRE:XXXXX CODIGO:XXXXX”.

2. Declare una nueva clase llamada “PRODUCTO_STOCK” que **herede** todos los miembros de la primera, adicionando dos miembros privados ***unsigned int Cantidad*** y ***char* Proveedor*** que deberán amanecer en cero y “SIN PROVEEDOR” al ser instanciada la clase sin parámetros.

3. Desarrolle los constructores parametrizados y destructores de la clase base y derivada, reutilizando código cuando sea posible.

4. Agregue otros dos métodos operator++ y operator— que sumen o resten la cantidad de los productos en el stock. Deben seguir la lógica de ***pos incremento*** y ***pos decremento***.

5. Desarrolle algún mecanismo dentro de la clase ***PRODUCTO_STOCK*** para saber la cantidad de productos que se tienen en total.

6. Realice un pequeño programa que utilice todos los métodos de la clase ***PRODUCTO_STOCK*** como ejemplo.

IPCs:

1. Dada la siguiente clase, realizada para implementar un Message Queue:

```
class IPC
{

protected:
    key_t llave;
    bool borrar;
    bool generateKey(char * p = nullptr , char a = 0);

public:

    IPC(char * p = nullptr , char a = 0 , bool destroy = false);
    key_t   getKey  (void) const;
    void    borrarAlFinal   (bool);
};
```

```
class MsgQueue : public IPC
{
private:
    struct mymsgbuf{
        long mtype;
        char mtext[MAX];
    };

    int id;

public:

    static const int DISCONNECTED = -1;

    MsgQueue(char * p = nullptr , char a = 0 , bool destroy = false);
    MsgQueue(const MsgQueue & a);
    bool    conectar( char * , char );
    int     getID   (void) const;
    bool    enviarMsj  ( void * , int , long typ = 1) const;
    int     recibirMsj ( void * , int , long typ = 0) const;
     ~MsgQueue();

};
```

Se pide:

a. Desarrollar el método ***bool enviarMsj (void *, int , long)***, que recibe como parámetros:
   - Un puntero al mensaje que se desea enviar.
   - Un entero que indica el tamaño de dicho mensaje, en bytes.
   - Un long int, que indica el tipo de mensaje que se desea enviar.

   Dicho método deberá poner el mensaje en la cola de mensajes, siempre y cuando los valores de tamaño del mensaje y de tipo sean mayores a 0, y siempre que la cola se encuentre correctamente inicializada (en cuyo caso ID será distinto a ***MsgQueue::DISCONNECTED***).

   La función devolverá true si pudo realizar la operación correctamente, o false en caso contrario.

b. Sobrecargar el operador << de manera de poder enviar distintos mensajes, haciendo uso de la función realizada en el punto anterior. Se desea poder realizar las siguientes operaciones:

```
int variable = 3;
MsgQueue Cola(".",'a');
if ( Cola.getID() != MsgQueue::DISCONNECTED )   {
   Cola<<"El valor de la variable entera es: ";
   Cola<<variable;
}
```

   Nota: Los mensajes enviados con el operador << deberán tener el tipo 1.