INFORMATICA II	SUBTEMA	TEMA #	PARCIAL	26/06/2021
	C++/IPC	2	# 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++:

Para un sistema de gestión de vuelos, se cuenta con las siguientes clases:

```
class Avion {
private:
    String Matricula,  //XB218 por ejemplo
    String Cia;     //compañía aérea: Austral, LATAM, etc
public:
    //constructor parametrizado con valores iniciales triviales a su elección
    //constructor de copia
    ~Avion();
    // agregar mas métodos si considera que hace falta
};
```

```
class Hora {
    int hora, min;
public:
    Hora();   //inicializa en 0,0
    Hora(Hora&);
    //pre incremento
   // pos decremento
};
```

```
class String
{
private:
    char *p;
    short tam;
public;
    String();
    String(const char *);
    String(const String&);
    ~String();
            operator= (const String&);
           operator= (const char *);
            operator+= (const char *);
           operator+ (const String&);
           operator+= (const String&);
    friend String operator+ (const char*,  const String&);
    friend ostream& operator<< (const ostream&, const String&);
    friend istream& operator>> (const istream&, const String&);
};
```

Se pide

1) Desarrollar los siguientes métodos de la clase **String**

? operator+ (const String&)

? operator+= (const char*)

2) En la clase **Avion**:

Declarar y desarrollar los métodos señalados por los comentarios en rojo.

3) Completar la clase **Hora** con los métodos señalados por los comentarios en rojo.

4) A partir de la clase **Avion**, heredar una nueva clase **Partida**, que debe contar con un objeto de tipo String llamado *Destino* (para la ciudad destino del vuelo), otro de tipo int llamado *Puerta* (puerta de embarque del vuelo) y un objeto de tipo Hora. Además, debe incluir y desarrollar los siguientes métodos:

- Constructor por defecto con valores triviales arbitrarios elegidos por Ud.
- Constructor de copia.
- Constructor parametrizado que incluya un valor por defecto para el destino (“a definir”) y asegure la correcta inicialización de los miembros de la clase Avion y Hora..

NOTA: Todos estos constructores deben inicializar correctamente todas las variables en juego (de Avion, de Partida y de Hora)

- void set_destino (String)
- sobrecarga del operador << para imprimir por pantalla los datos completos de las partidas utilizando convenientemente los métodos disponibles.

5) Realice un main() que instancie en tres oportunidades la clase **Partida** utilizando cada uno de los constructores creados. Imprima un listado de todos los datos involucrados en cada objeto **Partida** utilizando convenientemente y sin redundancias las codificaciones realizadas previamente.

IPCs:

Dada la siguiente clase, realizada para implementar una Shared Memory:

```
class IPC
{

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

public:

    static const int DISCONNECTED = -1;

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

```
class ShMem : public IPC
{
private:
    int id_shmem, id_sem;
    char *buffer;
    int size;

    void bloquearShMem();
    void desbloquearShMem();

public:
    ShMem   (char * p = nullptr , char a = 0, int tam = TAM , bool destroy = false);
    ShMem   (const ShMem &);

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

    int     getID   (void) const;
    int     getSize (void) const;

    bool    escribirMensaje ( void * msj , int cant , int pos = 0 );
    int     leerMensaje ( void * buf , int pos = 0 ,  int cant = TAM );

    ~ShMem();
};
```

Se pide:

a. Desarrollar el método ***bool conectar (char *, char , int)***, que recibe como parámetros:
   - Un puntero a char, para albergar una cadena de caracteres que hace referencia a un nodo del filesystem.
   - Una variable de tipo char.
   - Un entero, que indica el tamaño de la shared memory a generar.

   Dicho método deberá verificar que la shared memory no se encuentre creada (en cuyo caso ID será distinto a ***IPC::DISCONNECTED***), y de comprobarse este punto deberá generar una llave a partir de los 2 primeros parámetros recibidos y crear y conectarse a una shared memory a partir de esta llave. Adicionalmente, se deberá crear y conectarse a un semáforo para controlar la escritura y lectura de la misma, utilizando la misma llave. Las variables privadas ***llave, id_shmem, buffer*** y ***id_sem*** deberán quedar cargadas con los valores devueltos por dichas funciones. De producirse algún error, la función devolverá *false*, y id_shmem quedará cargado con el valor ***IPC::DISCONNECTED***). En caso contrario el método devolverá *true*..

b. Realizar el destructor de la clase ShMem, que deberá eliminar los IPCs en caso de que la variable booleana ***borrar*** sea true.