UNIVERSIDAD DE MÁLAGA

E.T.S.I. TELECOMUNICACIÓN

Dpto. de Lenguajes y Ciencias de la Computación

Programación 2
(1° de Grados de SE, SI, ST, TT y Tel)
Septiembre 2017

Apellidos, Nombre	*Ord*	*Grado*	*Grupo*	*Hora Entrega*

ESTE ENUNCIADO DEBE ENTREGARSE AL PROFESOR

Instrucciones para la realización del examen:

- El código fuente de cada ejercicio debe guardarse en los ficheros con los nombres siguientes: `proceso.hpp`, `proceso.cpp`, `cola_procesos.hpp`, `cola_procesos.cpp`, `planificador.hpp`, `planificador.cpp` y `main_planificador.cpp`.
- Estos ficheros deberán comenzar con un comentario con el nombre, apellidos, ordenador, grado y grupo.
- Se deben respetar en todo momento los nombres especificados en el enunciado (nombres de ficheros, identificadores de tipos y de funciones miembro), así como los prototipos para las funciones requeridas.
- Los programas resultantes deberán compilar correctamente y funcionar según lo especificado en el enunciado.
- Se valorará el código bien estructurado.
- **Se debe obtener una calificación superior ($\geq$) a 3 puntos en el ejercicio 2 (TAD `Planificador`) para que el resto de ejercicios puedan ser calificados.**
- Al finalizar el examen:
  - Se deberán subir los ficheros `proceso.hpp`, `proceso.cpp`, `cola_procesos.hpp`, `cola_procesos.cpp`, `planificador.hpp`, `planificador.cpp`, y `main_planificador.cpp` a la tarea especificada del *Campus Virtual* de la asignatura.
  - Además (salvo en los Mac), se deberán copiar los archivos `proceso.hpp`, `proceso.cpp`, `cola_procesos.hpp`, `cola_procesos.cpp`, `planificador.hpp`, `planificador.cpp`, y `main_planificador.cpp` al directorio de **Windows** [`Documentos\examen\p2sep17\`] (si no existe la ruta, deberá crearse).
  - Se deberá escribir la *hora de entrega* (además del resto de datos personales) en la cabecera del enunciado del examen y entregar al profesor.

Con objeto de gestionar los procesos que deben ser ejecutados por un micro-procesador en un ordenador, se desea desarrollar una aplicación para la gestión de diferentes colas de prioridad para encolar los procesos antes de ser ejecutados. Para ello, la aplicación requiere el desarrollo de las clases `Proceso` (para la gestión de un único proceso), `ColaProcesos` (para la gestión de una cola de procesos) y `Planificador` (para la gestión de una lista de colas de procesos, cada una con una prioridad diferente)

En el campus virtual de la asignatura se suministran los ficheros fuente `proceso.hpp` y `proceso.cpp`, que implementan la clase `Proceso`, desarrollada en el espacio de nombres `bblProgII`. Esta clase encapsula los datos (`identificador` del proceso, su `prioridad`, la cantidad de `memoria` y el `tiempo_cpu` necesario para su ejecución) y suministra las operaciones necesarias para gestionar un único proceso.

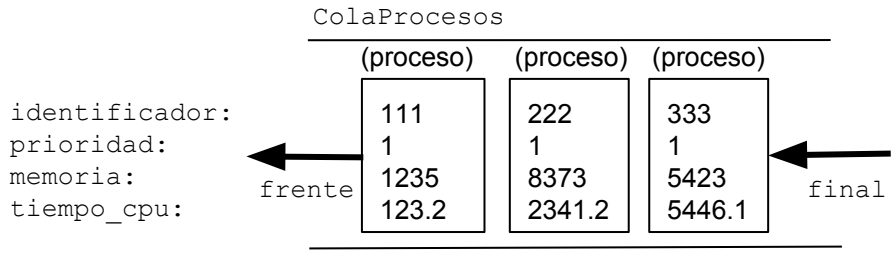
La clase `Proceso` suministra los siguientes métodos públicos:

```
// Constructor por defecto.
Proceso();

// Constructor que inicializa el proceso con todos sus datos (id, prioridad,
// memoria y tiempo_cpu).
Proceso(unsigned id, unsigned pri,
        unsigned mem, double t_cpu);

// Constructor de copia de la clase.
Proceso(const Proceso &p);

// Consulta el id del proceso.
unsigned consultar_id() const;

// Consulta la prioridad del proceso.
unsigned consultar_prioridad() const;
```

```
// Consulta la memoria empleada por el proceso.
double consultar_memoria() const;

// Consulta el tiempo de cpu empleado por el proceso en su ejecución.
double consultar_tiempo_cpu() const;
```

Ejercicio 1 - La clase `ColaProcesos` *(3 pts)*

Una cola de procesos es una estructura de tipo FIFO (el primero que entra es el primero que sale) que permite encolar procesos (objetos de tipo `Proceso`) y suministra las operaciones para insertar (por el final), consultar y eliminar (por el frente) de la cola (véase la figura, que representa una cola con 3 procesos, todos de prioridad 1).



Un objeto de la clase `ColaProcesos` debe encapsular (atributos) un *array* de procesos de tamaño `MAX_NUM_PROCESOS` (constante simbólica definida en el espacio de nombres `bblProgII` o como constante estática privada), junto con el número de procesos (`num_procesos`) actualmente encolados en la cola.

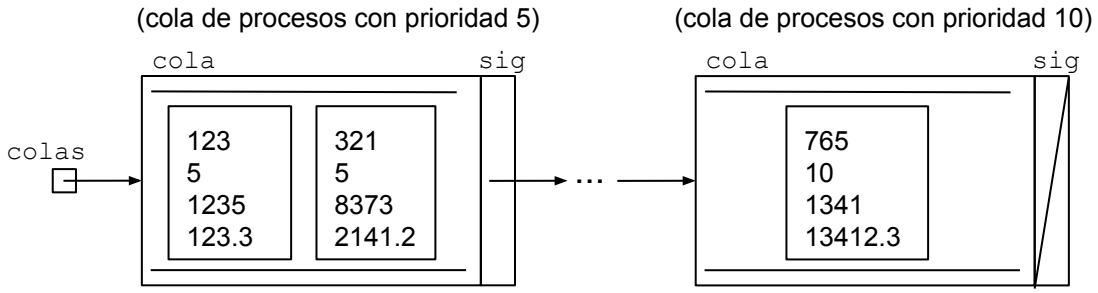
Además, la clase `ColaProcesos` debe suministrar los siguientes métodos públicos:

```
// Inicializa la cola de procesos a cola vacía.
ColaProcesos();

// Constructor de copia de la clase.
ColaProcesos(const ColaProcesos &c);

// Indica si la cola está o no vacía.
bool colaVacia() const;

// Indica si la cola está o no llena.
bool colaLlena() const;

// Devuelve el número de procesos
// encolados en la cola
unsigned numProcesos() const;

// Consulta el frente (primer elmento) de la cola.
// PRECONDICIÓN: la cola no está vacía.
void frente(Proceso &p) const;

// Encola (por el final) un proceso en la cola.
// PRECONDICIÓN: la cola no está llena.
void encolar(const Proceso &p);

// Desencola (elimina por el frente) un proceso.
// PRECONDICIÓN: la cola no está vacía.
void desencolar();

// Indica si un proceso, con identificador 'proc_id',
// está almacenado (true) o no (false) en la cola.
bool estaEnCola(unsigned proc_id) const;
```

Ejercicio 2 - La clase `Planificador` *(7.0 pts)*

La clase `Planificador` permite gestionar una lista de colas de procesos, cada una de las cuales almacena solo procesos de una misma prioridad. La lista de colas, implementada como una **lista enlazada lineal simple** de objetos `ColaProcesos`, está ordenada en orden creciente de prioridad de los procesos encolados en cada cola (véase la figura, donde se muestran el primer y el último nodo de la lista, con las colas de procesos de prioridades 5 y 10: la primera con dos procesos encolados y la última con solo un proceso en cola).

`Planificador`



La clase `Planificador` debe suministrar los siguientes métodos públicos:

```
// Constructor por defecto: lista de colas vacía.
Planificador();

// Constructor de copia.
Planificador(const Planificador &p);

// Destructor.
~Planificador();

// Si el identificador del proceso no existe en la cola correspondiente a su prioridad,
// el proceso se encola en dicha cola. Si la cola para esa prioridad no existe,
// se crea esa nueva cola (nodo) en el planificador y el proceso se encola en esa nueva cola.
// La nueva cola debe insertarse en orden creciente de prioridad de los procesos que esa cola
// guarda (siendo 0 la máxima prioridad). Si la cola de procesos para esa prioridad ya
// existe, el proceso se encola en ella. Si el proceso se ha podido encolar, se devuelve true
// a través de 'exito'. Si no, se devuelve false.
void encolar(const Proceso &p, bool &exito);

// Si hay colas de prioridad, se desencola el primer proceso de la cola de máxima prioridad
// (la primera en la lista de colas) y se devuelve el identificador del proceso desencolado
// a través del parámetro 'id_desencolado', además de true a través del parámetro 'exito'.
// Si la cola de procesos de donde se ha desencolado el proceso queda vacía,
// debe eliminarse de la lista de colas.
// Si la lista de colas estaba vacía, debe devolverse false a través de 'exito'.
void desencolar(unsigned &id_desencolado, bool &exito);

// Devuelve el número de colas que hay en este momento en el planificador.
unsigned num_colas() const;

// Devuelve un vector con el número total de procesos encolados en
// cada una de las colas de prioridad del planificador.
std::vector<unsigned> num_procesos() const;

// Busca el proceso cuyo identificador se pasa como parámetro en la cola de procesos del
// planificador correspondiente a su prioridad. Si el proceso existe en el planificador,
// debe devolverse a través del parámetro 'p'; además, se devuelve true a través de 'existe'.
// Si el proceso no está en el planificador, se devuelve false.
void consultar_proceso(unsigned id_proceso, Proceso &p, bool &existe) const;

// Escribe en pantalla la información de todos los procesos encolados en las colas
// del planificador, en orden de prioridad.
void escribir() const;

// Lee del fichero que se pasa como parámetro la información de una serie de
// procesos que deben encolarse en el planificador conforme se leen.
// El formato del fichero es:
//  id_proceso_1
//  prioridad_1
//  memoria_proceso_1
//  tiempo_cpu_proceso_1
//  id_proceso_2
//  prioridad_2
//  memoria_proceso_2
//  tiempo_cpu_proceso_2
// ...
void leer_fichero(const std::string &nom_fic);
```

En el campus virtual de la asignatura se suministra el fichero fuente `main_planificador.cpp`, que contiene el código necesario para probar los distintos métodos de la clase `Planificador`.