PRÁCTICA DE LA ASIGNATURA
LABORATORIO DE PROGRAMACIÓN II
Curso 2008/09

Ingeniería Informática
Ingeniería Técnica en Informática de Sistemas
Ingeniería Técnica en Informática de Gestión

Roberto Rodríguez Echeverría (rre@unex.es)
Encarna Sosa Sánchez (esosa@unex.es)
José María Conejero (chemacm@unex.es)

PRISONBREAK REDUX

Entrega 1

El proyecto de simulación de la prisión será realizado mediante un proceso iterativo e incremental. Para ello, la dirección del proyecto ha dividido el mismo en tres entregables diferentes.

En este documento se especifican las acciones que deben llevarse a cabo para implementar la primera entrega del proyecto. Esta primera entrega del proyecto se centra en el análisis, implementación y prueba de las puertas existentes en cada planta. Por tanto, a continuación se detalla el funcionamiento de dichas puertas y las características de las llaves que se utilizarán para abrirlas.

La puerta

Cada puerta cuenta con una cerradura de seguridad que los presos intentarán abrir. Esta cerradura puede presentar tres estados diferentes: “no configurada”, “cerrada” o “abierta”. La cerradura dispone de una combinación secreta que debe ser configurada inicialmente antes de poder usarla. Una vez configurada, debe cerrarse la puerta. Sólo es posible abrirla usando la combinación de llaves adecuada que sea capaz de cambiar su estado a abierta. El cambio de estado dependerá de que se cumpla o no la condición de apertura. Una vez abierta, la puerta puede volver a cerrarse. Al cerrarse volverá a establecerse la combinación que se configuró inicialmente. La figura 1 muestra un diagrama de estados UML que especifica el comportamiento de la puerta.

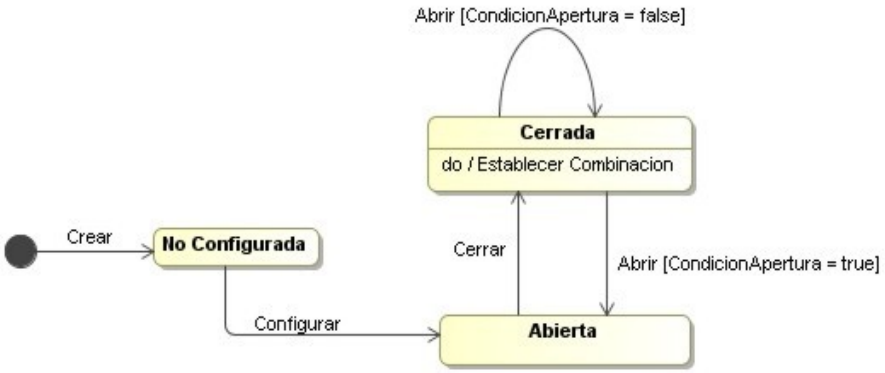

Figura 1. Diagrama de estados UML que representa el funcionamiento de la puerta

La combinación secreta de la cerradura está formada por un conjunto de llaves. Para abrir la puerta, se debe usar un conjunto de llaves, probadas de una en una, hasta que se acierte con la combinación secreta. Al probar una llave, la puerta comprueba si la llave forma parte de la combinación secreta y, en este caso, la elimina de la misma. Las búsquedas de las llaves en la combinación secreta de la cerradura deben ser lo más eficientes posibles.

La condición de apertura se basa en que se cumplan dos subcondiciones. Primero, la profundidad de la combinación debe ser menor de un valor predeterminado, suponiendo que se trata de una estructura balanceada. Y, segundo, que la combinación tenga mayor número de llaves internas que finales.

Además, la cerradura tiene un sistema de alarma que se activa cuando se intenta probar una misma llave dos veces. Por esta razón, antes de comprobar si una llave abre la cerradura, se debe comprobar si la llave ha sido probada anteriormente en la cerradura. Para realizar esta acción, la puerta dispone de otra memoria optimizada para la búsqueda de códigos en la que se van almacenando aquellas llaves que ya han sido probadas. Esta memoria se vacía cada vez que se cierra la puerta; ya que, la puerta debe volver a su estado inicial.

Las llaves

Cada llave dispondrá de un código numérico que permita identificarla, aunque podrán existir varias llaves con el mismo código. Además, las llaves son capaces de almacenar información biométrica del último individuo que la utilizó, permitiendo identificar al mismo en el estudio de análisis final de la simulación.

Las combinación secreta de llaves para una puerta será generada al comienzo de la simulación. Para cada planta de la prisión se generarán 30 llaves, cuyos identificadores irán de 0 a 29. De todas estas llaves, las que tengan identificador impar se utilizarán para configurar la puerta de esa planta: {1,3,5,7,9,11,13,15,17,19,21,23,25,27,29}

Además, las llaves con identificador impar generadas para una planta serán duplicadas, con el objetivo de que en una misma planta puedan existir llaves repetidas. De este modo, para cada planta, se dispondrá de un conjunto de llaves (ordenado por identificador) como el que sigue:
{0,1,1,2,3,3,4,5,5,6,7,7,8,9,9,10,11,11,12,13,13,14,15,15,16,17,17,18,19,19,20,21,21,22,23,23,24,25,25,26,27,27,28,29,29}

Dado que en esta primera entrega se pretende simular el funcionamiento de

una puerta, se debe realizar la carga inicial de la cerradura con la combinación de llaves indicada más arriba e intentar abrir la cerradura probando el conjunto de llaves que existirían en la planta (conjunto de llaves con identificadores impares repetidos).

Simulación.

En esta primera entrega, las acciones a simular son las siguientes:

- Generación de la combinación secreta de la puerta de una planta.
- Configuración de la puerta con la combinación secreta.
- Duplicación de las llaves con identificador impar.
- Intento de apertura de la puerta, probando las diferentes llaves.
- Cerrar la puerta: vuelta al estado inicial.

PRIMERA PRUEBA DE LA EVALUACIÓN CONTINUA (EC1)

De cara a la realización final de la práctica, la primera prueba consistirá en una modificación sobre el comportamiento inicial de la puerta de una planta. La modificación se centrará en la realización de algoritmos sobre un árbol binario de búsqueda de llaves, debiéndose presentar un proyecto de programación que contenga lo siguiente:

- La implementación de una clase “Puerta” que cumpla los requisitos descritos en este documento.
- La implementación de una clase “Llave” que cumpla los requisitos descritos en este documento.
- La implementación de la estructura de datos árbol binario de búsqueda (abb). Las operaciones básicas del árbol a implementar son las típicas de esta estructura de datos, incluyendo sus correspondientes recorridos y, además, las operaciones de equilibrado necesarias para comprobar la condición de apertura de la puerta. Para esta estructura de datos se facilitará una implementación por parte de los profesores de la asignatura. Es muy recomendable utilizar esta implementación en la práctica.
- La implementación y utilización de, al menos, una de las estructuras de datos lineales (lista, pila y cola) comentadas en las sesiones prácticas. Estas estructuras de datos deben haber sido implementadas por el alumno y no pertenecer a la librería STL. Además, deben ser genéricas, de modo que puedan ser utilizadas con diferentes tipos de objetos sin necesidad de cambiar su implementación. Incluso, estas estructuras de datos deben permitir almacenar objetos estáticos o dinámicos sin necesidad de cambiar su implementación.
- Para la realización de ciertos algoritmos de esta fase del proyecto, se permitirá la utilización de estructuras de datos de la STL.

La implementación debe seguir los conceptos de Programación Orientada a Objetos vistos en clase y debe gestionar correctamente la memoria dinámica. Además, se considera necesario conocer y comprender los conceptos de sobrecarga.

Cada alumno deberá realizar la entrega de esta primera fase del proyecto antes de presentarse a la modificación. Para realizar esta entrega se habilitará una tarea en el aula virtual. Posteriormente, el alumno realizará la prueba de modificación sobre el mismo código que haya entregado en el aula virtual.

Requisitos sobre el código de la EC1:

- El código debe estar perfectamente documentado con la notación doxygen y debe generar la documentación correctamente.
- El código debe estar escrito siguiendo un único estilo de programación.
- En la documentación interna del código debe indicarse:

  Nombre y Apellidos:
  Asignatura:
  Grupo:
  Número de Entrega:
  Curso:

Antes de la prueba se entregará:

- Una carpeta cuyo nombre sea igual al identificador de correo-e del alumno (sin "@alumnos.unex.es"). Dentro de esta carpeta se incluirán solamente los ficheros fuente que formen parte del ejecutable final **(*)**.

Al final de la prueba se entregará:

- Una carpeta cuyo nombre sea igual al identificador de correo-e del alumno (sin "@alumnos.unex.es"). Dentro de esta carpeta se incluirán solamente los ficheros fuente que formen parte del ejecutable final **(*)**.

En ambos caso, la carpeta creada será comprimida en un fichero en formato **ZIP** con el mismo nombre. La entrega de dicho fichero comprimido se realizará a través de una actividad propuesta en Avuex, de forma que cada alumno subirá a la plataforma virtual dicho fichero. Es muy importante que cada alumno suba su archivo con su cuenta en Avuex.

(*) Nota: deben eliminarse de los proyectos entregados aquellos ficheros que no son necesarios para su corrección, ejemplo: ficheros objeto y fichero ejecutable.

A continuación se muestra un ejemplo de programa principal que simula las acciones mínimas que el proyecto entregado debe llevar a cabo. Este código es meramente orientativo.

EJEMPLO DE PROGRAMA PRINCIPAL

```
/**
* \file EC1 - proyecto08_09
* \brief Implementacion de la EC1 dentro del proyecto del curso 08-09
* \author
*   \b Profesores LPII \n
*   \b Asignatura Laboratorio de Programacion II \n
*/

//! Constante NUMLLAVES que define el número de llaves que se generarán por planta
#define NUMLLAVES  30;

/**
 * Programa principal - EC1
 * \param argc Parametro con el numero de parametros que recibe el programa
 * \param argv Parametro con todos los parametros de tipo char* que recibe el programa
 * \return Retorna la salida del programa
 */
int main(int argc, char *argv[])
{
	//Crear estructuras de datos para almacenar las llaves creadas
	Estructura_Llaves combinacion, llaves_recogidas;

	// Generar la combinación secreta y la lista de llaves a probar
	for (int i=0; i<MAX_CELDAS; i++) {
		llaves_recogidas.insertar(new Llave(i));
		if ((i % 2) != 0) {
			llaves_recogidas.insertar(new Llave(i));
			combinacion.insertar(new Llave(i));
		}
	}

	//Crear una instancia de Puerta
	Puerta puerta;

	// Configurar la puerta cargando la combinación secreta
	puerta.configurar(combinacion);

	//cerrar la puerta (incialmente está abierta) – establece la combinación
	puerta.cerrar();

	// Mostrar estado de la puerta
	cout << puerta;

	bool abierta = false;
	// Intentar abrir la puerta
	Estructura_Llaves::iterator it = llaves_recogidas.begin();
	while ((!abierta) && (it!=llaves_recogidas.end())) {
		if (!puerta.llave_probada(*it))
			abierta=puerta.abrir(*it);
		it++;
	}

	if (abierta)
		cout << “¡Puerta abierta!” <<endl;
	else
		cout << “No se ha conseguido abrir la puerta” <<endl;

	// Mostrar estado de la puerta
	cout << puerta;

	// Cerrar la puerta – Establece la combinación de la puerta de nuevo
	puerta.cerrar();

	// Mostrar estado de la puerta
	cout << puerta;
```

```
        //INCLUIR AQUÍ EL CÓDIGO DE LA MODIFICACIÓN
        //************************************************


        //************************************************

        // Liberar la memoria dinámica usada
        // Comprobación de que la memoria se libera correctamente
        return EXIT_SUCCESS;
}
```