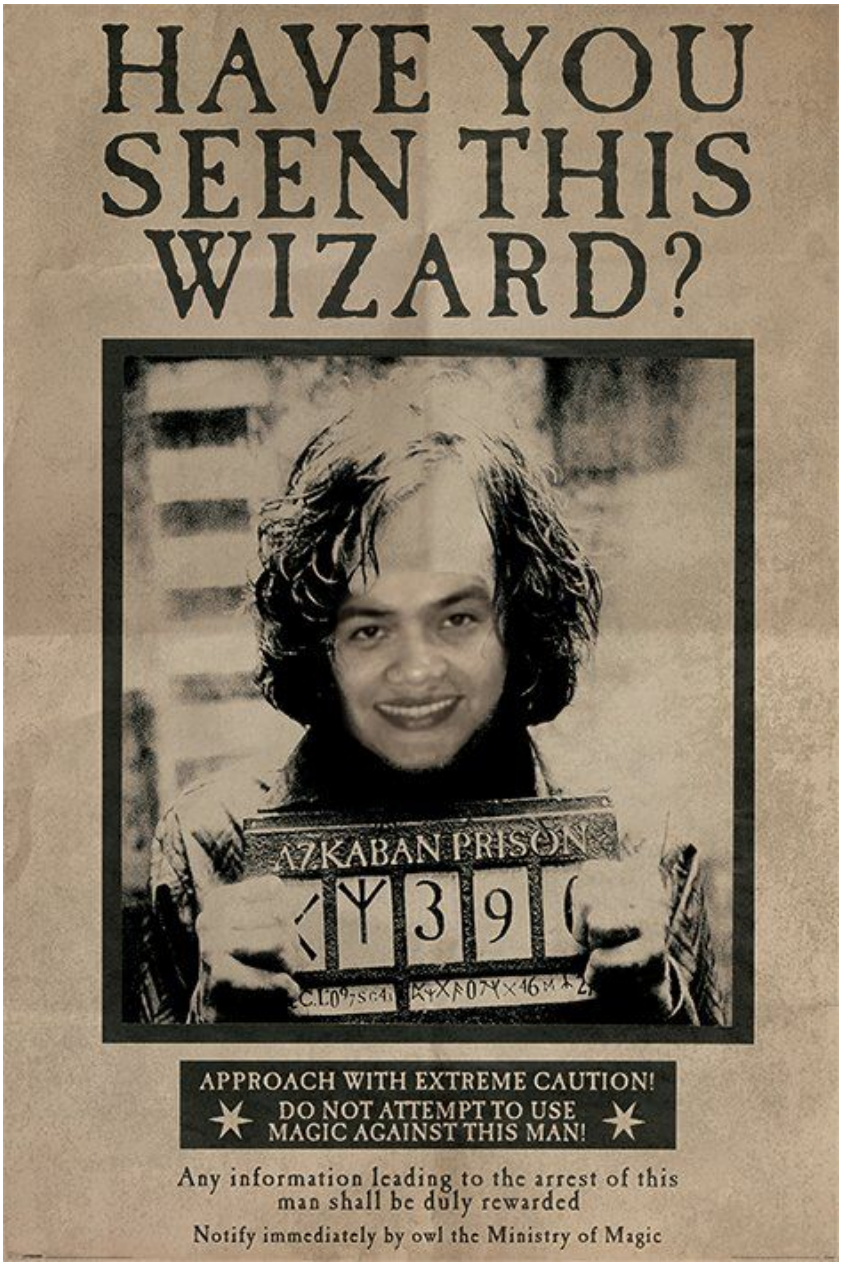
FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Algoritmos y Programación I
Curso Mendez

TP3: Prisión de Azkaban



Fecha presentación	06/06/2019
Fecha entrega	30/06/2019

Introducción

Azkaban es una prisión situada en una isla en el medio del Mar del Norte, que sirve a la mágica comunidad de Gran Bretaña. Con el uso de ciertos encantamientos, la prisión está oculta para el mundo muggle, y no puede ser localizada por uno de estos. Se cree que posee encantamientos para hacer la prisión más grande en su interior que lo que se ve en el exterior, ya que ésta sirve para la totalidad de los magos oscuros de Gran Bretaña. Desde 1717, el uso de una de las tres Maldiciones Imperdonables (Avada Kedavra) en otro ser humano, te lleva a cadena perpetua en Azkaban, mientras que las otras dos Maldiciones (Crucio e Imperius) imperdonables, pueden ser “perdonadas” si realizaste solo una de ellas.

Objetivo

- Que los alumnos se familiaricen con el manejo de archivos en C.
- Que los alumnos utilicen las funciones acordes al tipo de archivo y/o forma de acceso.
- Que los alumnos apliquen apropiadamente las operaciones con archivos si se dan las condiciones para ello

Enunciado

La prisión de Azkaban guarda la información de sus presos según la Maldición que usaron. Para esto **se cuenta con dos archivos binarios de acceso secuencial**, los cuales están ordenados por nombre, en uno de ellos figuran los presos que realizaron la maldición **Crucio** (crucio.dat) y en el otro los que realizaron la maldición **Imperius** (imperius.dat). No le interesan los que realizaron Avada Kedavra ya que esa se paga con cadena perpetua.

Para liberar a los presos, la prisión necesita saber quiénes cumplen las condiciones para poder ser perdonados. Los únicos que pueden ser alguna vez liberados son aquellos que hicieron sólo una de las dos Maldiciones “perdonables”.

Para esto, se debe crear un **archivo binario de acceso secuencial** en el cual figuren los presos que se encuentran en UNO de los dos archivos mencionados anteriormente (si un mismo preso aparece en los dos archivos, no podrá ser perdonado) ya que significa que realizó 2 maldiciones y no puede ser liberado.

Cada mes **se crea un archivo de texto** con los nombres de quienes van a ser liberados, pero OJO esto ocurre sólo si su **fecha de liberación coincide con el año/mes ingresado y su conducta es la requerida**.

Es necesario también que **se actualicen los dos archivos binarios** mencionados al principio, eliminando a todos los que fueron liberados.

Por último, se debe encontrar la forma de **mostrar los nombres** de todos aquellos que liberados en cierto año/mes.

Importante! No puede haber dos presos con mismo nombre, cada uno de ellos es único.

La administración le pidió a Harry que desarrolle un programa llamado **azkaban** que, tomando los argumentos recibidos en la línea de ejecución, realice una acción determinada. Como Harry es muy bueno en magia pero malo programando, nos pidió ayuda a nosotros para poder realizarlo. Las funciones son las siguientes:

- **Generar un archivo de presos “perdonables”.**

  Dados dos archivos binarios en los cuales figuran los presos dependiendo de la maldición que hayan realizado, se debe generar un archivo (también binario) de los presos que no hayan realizado ambas maldiciones. El nombre del nuevo archivo es recibido por parámetro.
- **Filtrar por fecha los presos a ser liberados en el año/mes.**

  Dado el archivo binario de los presos “perdonables”, se debe generar un archivo de texto con los nombres de quienes deben ser liberados ese año/mes y que, además, tienen buena conducta. El año/mes es recibido por parámetro.
- **Actualizar archivos.**

Dado el archivo de presos liberados, se deben actualizar los dos archivos binarios del principio, sacando de ellos los que fueron liberados. Se recibirá por parámetro el año/mes para saber que archivo de liberados utilizar.

- **Mostrar liberados.**

  Muestra por pantalla el archivo de presos a liberados de un determinado año/mes recibido por parámetro.

- **Mostrar ayuda sobre los comandos a utilizar.**

  Muestra por pantalla el listado de comandos y explica al usuario cómo se utilizan.

Requerimientos

El primer parámetro enviado a este programa debe ser el identificador de operación según se definirá a continuación:

- Comando **perdonables**:

  De dos archivos con los presos que realizaron cada maldición, crear uno con aquellos presos que pueden ser perdonados. El segundo argumento enviado será el nombre del archivo a crear.

  - Comando identificador: perdonables
  - Argumentos:
    - nombre del archivo a crear. El usuario decide el nombre.
  - Ejemplo:

    ./azkaban perdonables <nombre_archivo>

Al ejecutar esta línea, el programa deberá acceder a los dos archivos (crucio.dat e imperius.dat), y generar otro con aquellos a perdonar y con el nombre recibido como argumento. Si ese archivo existe, se sobreescribe.

- Comando **liberar**:

  Creará un archivo de texto con los nombres de los presos que cumplan con las condiciones necesarias.

  - Comando identificador: liberar
  - Argumentos:
    - Nombre del archivo de presos perdonables
    - año/mes (formato aaaamm)
    - conducta (B: buena - R: regular - M: mala)
  - Ejemplo de la línea de comando:

    ./azkaban liberar <archivo_perdonables>.dat 201906 B

Al ejecutar esta línea, el programa deberá acceder al archivo que se creó con el comando perdonables y generar otro archivo con los presos que deben ser liberados ese año/mes y que además tengan conducta mejor o igual a la recibida como argumento (B mejor que R mejor que M), el nombre de este archivo debe ser del tipo liberados**_aaaamm**.txt siendo **aaaamm** la fecha recibida como argumento.

Si ya existe un archivo para esa fecha, NO puede sobreescribirse. También se podrá liberar presos que tengan fecha de liberación pasada ya que en su momento podría no haberles tocado ser liberados por la conducta, por lo que si ahora tienen la conducta requerida pueden ser liberados.

- Comando **actualizar**:

  Actualizará ambos archivos binarios en los cuales figuran los presos que realizaron cada maldición.

  - Comando identificador: actualizar
  - Argumentos:
    - Fecha del archivo que se toma para actualizar
  - Ejemplo de la línea de comando:

    ./azkaban actualizar 201906

Al ejecutar esta línea, el programa deberá actualizar los archivos de presos crucio.dat e imperius.dat, sacando los presos que están en el archivo de los liberados.

- Comando **mostrar_liberados**:
  Mostrará por pantalla el archivo de presos liberados en un determinado año/mes. El segundo argumento será la fecha del archivo que se quiere mostrar.
  - Comando identificador: mostrar_liberados.
  - Argumentos:
    - Fecha del archivo que se quiere mostrar.
  - Ejemplo de la línea de comando:
    ./azkaban mostrar_liberados 201906

Al ejecutar esta línea, se debe mostrar por pantalla los nombres de los presos a ser liberados. Queda a criterio del alumno la forma de hacerlo.

- Ayuda
  - Comando identificador: ayuda.
  - Argumentos:
    - -
  - Ejemplo de la línea de comando:
    ./azkaban ayuda

Al ejecutar esta línea se debe mostrar una ayuda al usuario sobre cuales son las prestaciones del sistema y cómo se utiliza cada funcionalidad. Queda a criterio del alumno la forma en que se mostrará la información.

Archivos a utilizar

crucio.dat, imperius.dat y <archivo_perdonables>.dat

Todos son archivos binarios de acceso secuencial ordenados ascendentemente por nombre de los presos.

Los registros de estos archivos son del tipo **preso_t**.

```
#define MAX_NOMBRE 200
#define MAX_FECHA 7
typedef struct preso {
	char nombre[MAX_NOMBRE];
	unsigned int edad;
	char conducta; // B buena, R regular o M mala
	unsigned int pabellon;
	unsigned int celda;
	char fecha[MAX_FECHA]; //formato aaaamm
	int maldicion_realizada; //1 crucio, 2 imperius
} preso_t;
```

liberados_aaaamm.txt

Depende de la fecha, serán archivos de texto que **contienen solo los nombres de quienes vayan a ser liberados** en esa fecha. Ordenado por nombre.

Aclaraciones

En cada funcionalidad, cada archivo debe ser recorrido como máximo una vez. Se puede asumir que los archivos crucio.dat e imperius.dat están bien formados con la estructura indicada.

La ayuda también se debe mostrar por pantalla si alguno de los otros argumentos es ingresado erróneamente, o le faltan/sobran argumentos.

Se puede crear una biblioteca si lo considera pertinente, simplificador y cómodo, aunque no está obligado a hacerlo.

Aprobación

El trabajo debe poder ser compilado correctamente con la siguiente línea:

```
gcc *.c -o azkaban -std=c99 -Wall -Werror -Wconversion
```

Además se deben cumplir todas las pautas mencionadas y deben funcionar todas las opciones. Además de pasar las pruebas de Kwyjibo se tendrá en cuenta la legibilidad del código y las buenas prácticas al igual que en los trabajos anteriores.

Aclaración: Al poner *.c el compilador toma todos los .c de la carpeta y los compila. Por ende, no debería importar el nombre que le pongan a la biblioteca ni al programa. Ante cualquier eventualidad informaremos oportunamente.

Anexo

Uso de parámetros: argv y argc

Para poder pasar parámetros a un programa a través de la línea de comandos nos valemos de la siguiente declaración de la función main:

```
int main(int argc, char *argv[]){..
```

Argc contiene la cantidad de argumentos recibidos por el programa, debemos considerar que siempre será el número de argumentos pasados más 1, ya que el primer argumento se reserva para contener el nombre del programa. Argv es un vector de strings que contiene los parámetros pasados en el mismo orden en que fueron escritos.

El siguiente programa muestra un ejemplo de cómo hacer uso de estos parámetros:

```
#include <stdio.h>
#include <string.h>
```

```c
#include <stdlib.h>

int main(int argc, char *argv[]){

	printf ("El programa recibió %i argumentos\n", argc);
	int i;
	for(i = 0; i < argc; i++) {
		printf("Parámetro %d: %s\n", i, argv[i]);
	}

	if (strcmp (argv[1], "saludar") == 0){
		printf ("Hola!\n");
	}
	else if (strcmp (argv[1], "sumar") == 0 && argc == 4){
		int sumando_1 =  atoi (argv [2]);
		int sumando_2 =  atoi (argv [3]);
		printf ("%i + %i = %i\n", sumando_1, sumando_2, sumando_1 +
sumando_2);
	}
	return 0;
}
```

Una vez compilado, sólo nos resta ingresar la línea de comando según lo que se quiera hacer, si ingresamos:

```
./ejemplo sumar 20 54
```

Siendo **ejemplo** el nombre del programa, lo que se muestra es:

```
El programa recibió 4 argumentos
Parámetro 0: ./ejemplo
Parámetro 1: sumar
Parámetro 2: 20
Parámetro 3: 54
20 + 54 = 74
```

Atoi: Array to integer

```c
int atoi(const char *nptr);
```

Esta función sirve para convertir la porción inicial de una cadena de caracteres apuntada por **nptr** en un entero.

Para usarla debemos incluir la biblioteca `<stdlib.h>`.