Algoritmos y Programación I
Curso Mendez

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Miniestadísticas

TP3

Fecha de Presentación: 14/11/2019

Fecha de Vencimiento: 28/11/2019

1. Introducción

A diferencia de los Elfos y los Hombres los enanos no fueron creados directamente por Ilúvatar sino por el vala Aulë tomando algún mineral de las profundidades de la tierra (no se sabe con certeza cual pero se sospecha que debió haber sido uno muy duro debido a la alta resistencia de estos seres) creó a los siete padres de los enanos que posteriormente darían lugar a toda la raza, sin embargo Aulë había creado en secreto a los enanos y al ser descubierto, arrepentido se ofreció a destruir a sus creaciones pero antes de hacerlo fue disculpado por Ilúvatar y se les permitió vivir con la única condición de que no despertaran antes que los Primeros Nacidos.

Como su nombre lo indica, son más bajos que los hombres con una altura de entre 120 y 150 centímetros, pese a ello son robustos, corpulentos y más fuertes y recios que el resto de las razas. Todos tienen barba, tanto hombres como mujeres, y el cortársela es la mayor vergüenza y ofensa que se les que les puede hacer, mereciendo el odio y rencor de toda su raza.

Son grandes conocedores de la minería y orfebrería agobiados por la larga labor de la búsqueda y extracción de los minerales subterráneos. Poseen una gran longevidad, llegando algunos a vivir alrededor de los 600 años.

2. Objetivo

El presente trabajo práctico tiene como objetivo que el alumno:

- Se familiaricen con el manejo de archivos en C.
- Utilicen las funciones acordes al tipo de archivo y/o forma de acceso.
- Apliquen apropiadamente las operaciones con archivos si se dan las condiciones para ello.
- Pongan en práctica el uso de argumentos por línea de comando.

Por supuesto, se requiere que el trabajo cumpla con las buenas prácticas de programación profesadas por la cátedra.

Se considerarán críticos la modularización, reutilización de código y la claridad del código.

3. Enunciado

La presidencia de la comunidad enanánica cuenta con un archivo donde guarda información de todos los enanos y necesita un sistema que permita algunas actualizaciones sobre él y también obtener cierta información correspondiente a las misiones que comunmente hace su pueblo.

Se pide, entonces, un sistema que cumpla los siguientes requerimientos:

- **Actualizar censo:** Al conseguir la información de los enanos que fueron a alguna misión, debe actualizarse el archivo del censo. Algunos enanos vuelven de la misión, pero lamentablemente otros no lo logran. Sin embargo, a veces en la expedición se encuentran con enanos que creían muertos y vuelven a la comunidad.
- **Elegir enanos para misión:** Dados ciertos parámetros, se deberán elegir los enanos qué participarán de la expedición, generando un archivo binario con la información de cada uno. No pueden elegirse enanos obreros.
- **Promover enanos:** La organización por rangos de los enanos permite escalar, si se cumplen ciertas condiciones. Un enano puede ser obrero, aprendiz, guerrero, líder, o general. Los obreros nunca irán a misiones.
- **Listar enanos de un rango:** Dado un determinado rango, se deben listar todos aquellos enanos que estén enrolados en él.

4. Especificaciones

4.1. Comandos

4.1.1. Actualizar censo

Recibirá como parámetro el nombre del archivo de la misión que tiene los datos para actualizar. Dejará el archivo del censo actualizado, sumándole 1 a las misiones de los enanos que volvieron y eliminando a aquellos que no volvieron. Tener en cuenta que a veces, enanos que se creían muertos, regresan en otras misiones trayendo gran alegría al vecindario.

Un ejemplo de este comando es:

```
.\miniestadisticas actualizar_censo espiar_a_smaug
```

Dentro del archivo con el resultado de la misión, los enanos que no volvieron no deberán estar en dicho archivo y los que se encontraron y crían perdidos, deberán darse de alta.

El comando deberá chequear la existencia del archivo enviado, en caso de no existir no deberán realizarse modificaciones sobre el censo.

4.1.2. Elegir enanos para misión

Recibirá como parámetros, la cantidad de guerreros, lideres y generales que deberán integrar la misión y además el nombre de la misión (separado por _). Creará un archivo binario cuyo nombre será el ingresado o mision.dat si no se ingresa. El orden de los parámetros debe respetarse.

Siempre tienen que ir 10 aprendices a todas las misiones, a modo entrenamiento. Si no hay 10 enanos aprendices, se llevará los que haya disponibles.

Un ejemplo de este comando es:

```
.\miniestadisticas integrantes_mision 50 6 2 espiar_a_smaug
```

En este ejemplo, se solicitan 50 guerreros, 6 líderes y 2 generales. Si no se cuenta con los enanos correspondientes de cada rango el archivo deberá ser borrado. Si ya existe un archivo con el nombre ingresado, no debe llevarse a cabo. No podrán seleccionarse enanos obreros.

4.1.3. Promover enanos

No recibirá parámetros. Debe promover a aquellos enanos que cumplan con las condiciones necesarias para crecer dentro de la organización. Los criterios son:

- Para pasar de aprendiz a guerrero, tiene que tener más de 10 misiones.
- Para pasar de guerrero a líder, tiene que tener más de 100 misiones, por lo menos 40 años y menos de 50.
- Para pasar de líder a general, tiene que tener más de 250 misiones por lo menos 50 años y menos de 60.

Un ejemplo de este comando es:

```
.\miniestadisticas promover_enanos
```

Dará como resultado, el archivo censo actualizado. No podrán promoverse enanos obreros ni muertos.

4.1.4. Listar enanos de un rango

Recibirá como parámetro el rango que se desea listar. Deberá mostrar por pantalla como encabezado el nombre y descripción del rango y luego los datos de cada enano que pertenece a ese rango.

Un ejemplo de este comando es:

```
.\miniestadisticas listar_enanos 3
```

Listará todos los enanos guerreros.

4.2. Estructuras

```
typedef struct enano {
    char nombre[MAX_NOMBRE];
    int edad;
    int cantidad_misiones;
    int id_rango;
} enano_t;

typedef struct rango {
    int id;
    char nombre[MAX_RANGO]
    char descripcion[MAX_DESCRIPCION];
} rango_t;
```

Los tamaños de los vectores de dicha estructura son:

```
  MAX_NOMBRE 50
  MAX_RANGO 50
  MAX_DESCRIPCION 200
```

4.3. Archivos

4.3.1. censo.csv

Este archivo contendrá información de todos los enanos que pertenecen a la comunicad.

Es un archivo de texto, donde cada línea corresponde a un enano, sus campos están separados por ; y está ordenado ascendentemente por nombre de enano.

Un ejemplo de este archivo se muestra a continuación:

```
Balin;103;200;3
Bifur;40;30;3
Bofur;56;5;2
Bombur;38;36;3
Gimli;43;101;4
Thorin;54;300;5
...
```

4.3.2. rangos.dat

Este archivo contendrá información de todos los rangos, en particular, su nombre y descripción.

Es un archivo binario de acceso aleatorio, ordenado por id de rango, los ids son consecutivos, comienzan en 1 y no hay agujeros de ids.

4.3.3. mision.dat

Este archivo contendrá toda la información de todos los enanos que fueron seleccionados para participar de la misión. Si por algún motivo un enano se perdió o murió durante el combate, será indicado con un -1 en el campo edad de este mismo archivo.

El nombre del archivo estará conformado por la palabra mision y luego el nombre ingresado como parámetro, si no se ingreso nombre, deberá llamarse solamente mision.dat.

Es un archivo binario de acceso secuencial, ordenado por nombre de enano alfabéticamente.

Siempre tendrán extensión .dat aunque no se los ingrese en los comandos.

4.4. Convenciones

Se deberá tomar como convención que los ids de los rangos de los enanos son:

- 1: Obrero.
- 2: Aprendiz.
- 3: Guerrero.
- 4: Líder.
- 5: General.

5. Compilación y Entrega

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

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

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.

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

6. Anexo

6.1. Argumentos por línea de comando

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>
#include <stdlib.h>

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

        printf ("El programa recibió %i argumentos\n", argc);
        for(int 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
```

6.2. Atoi: Array to integer

```
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**.