Algoritmos y Programación II (75.41)
Trabajo Práctico Nº1

16 de septiembre de 2013

1. Introducción

El jefe de sistemas de una prestigiosa facultad ha decidido cambiar el sistema de inscripción a materias por uno más robusto. Aunque no tiene mucha idea del estado del arte de la programación, exige usar para este proyecto los más modernos frameworks que él conoce: Pilas, Colas, Listas y lo último en Vectores Dinámicos. Además, le comentaron que el mejor momento para ponerlo en línea es ahora, a una semana de la inscripción.

Dado que el programa correrá sobre el único servidor que posee esta casa de estudios, adquirido en 1998, y compartido por todos los otros servicios online de la facultad, es requisito que el sistema no desperdicie memoria durante su ejecución.

El nuevo sistema se llamará "Sunga Iraní", nombre inspirado en otro exitosísimo sistema de gestión académica. El jefe de sistemas lo contactó a usted para desarrollar este programa, con todas las características descriptas.

2. Consigna

Implementar en C el sistema de gestión de inscripciones a los cursos de la facultad, que debe proveer las siguientes operaciones:

- Operaciones de administración:
  - Agregar curso
  - Inscribir un alumno a un curso
  - Desinscribir un alumno de un curso
  - Eliminar curso
  - Listar inscriptos de un curso
- Servicios para alumnos:

- Listar cursos disponibles
- Iniciar sesión
- Ver el estado de la sesión
- Inscribirse a un curso
- Deshacer la última inscripción
- Cerrar y aplicar sesión

2.1. Protocolo

Este programa formará parte de un sistema automatizado. En lugar de presentar un menú con opciones en la pantalla, se esperará que el programa respete un *protocolo de comunicación*, en el cual se reciben comandos por la entrada estándar (`stdin`). Cada comando recibido debe ser procesado y su resultado debe ser escrito en la salida estándar (`stdout`).

Cada línea de la entrada corresponde a un comando, y todos los comandos respetan el mismo formato. El nombre del comando está separado de los parámetros (si existen) por un espacio, y los parámetros están separados por comas:

```
comando parametro1,parametro2,parametro3,...
```

2.2. Comandos de administración

2.2.1. Comando: `agregar_curso`

Formato: `agregar_curso idc,descripcion,materia,vacantes`

Descripción: Agrega un nuevo curso a la base de datos.

Parámetros:

`idc`: Identificador del curso. Cadena de texto de longitud no mayor a 10 caracteres.

`descripcion`: Nombre descriptivo del curso. Cadena de texto de longitud no mayor a 80 caracteres.

`materia`: Código de la materia. Cadena de texto de longitud no mayor a 10 caracteres.

`vacantes`: Cantidad de vacantes del curso.

Salida: `OK` en caso de no producirse error. Mensaje de error apropiado en caso contrario (ver sección 2.4).

Ejemplo:

```
agregar_curso a2rw,Algoritmos II - Rosita ↩
    ↪Wachenchauzer,75.41,60
OK
```

2.2.2. Comando: `inscribir`

Formato: `inscribir padron,idc`

Descripción: Inscribe un alumno en un curso.[1]

Parámetros:

`padron`: Número de padrón del alumno.

`idc`: Identificador del curso.

Salida: Mensaje apropiado en caso de error (ver sección 2.4). En caso de no producirse error debe informarse si el alumno quedó en condición regular o condicional:

```
OK: padron <padron> inscripto como <regular|condicional>
```

Ejemplo:

```
inscribir 55555,a2rw
OK: padron 55555 inscripto como regular
```

2.2.3. Comando: `eliminar_curso`

Formato: `eliminar_curso idc`

Descripción: Elimina un curso de la base de datos.

Parámetros:

`idc`: Identificador del curso.

Salida: `OK` en caso de no producirse error. Mensaje de error apropiado en caso contrario (ver sección 2.4).

Ejemplo:

[1] Cualquier alumno se puede inscribir en cualquier curso, independientemente de las vacantes. Si hay más alumnos que las n vacantes en un curso, los primeros n alumnos inscriptos quedan en condición regular, y el resto como condicionales.

```
eliminar_curso a2rw
OK
```

2.2.4. Comando: `desinscribir`

Formato: `desinscribir padron,idc`

Descripción: Elimina un alumno de un curso.

Parámetros:

`padron`: Número de padrón del alumno.

`idc`: Identificador del curso.

Salida: `OK` en caso de no producirse error. Mensaje de error apropiado en caso contrario (ver sección 2.4).

Ejemplo:

```
desinscribir 55555,a2rw
OK
```

2.2.5. Comando: `listar_inscriptos`

Formato: `listar_inscriptos idc`

Descripción: Muestra un listado de padrones inscriptos a un curso.

Parámetros:

`idc`: Identificador del curso.

Salida: Mensaje apropiado en caso de error (ver sección 2.4). En caso contrario, un alumno por línea, con el siguiente formato:

```
<padron> <regular|condicional>
```

Ejemplo:

```
listar_inscriptos
55555 regular
66666 condicional
```

2.3. Servicios para los alumnos

Por cuestiones de seguridad, los alumnos no tendrán acceso a los comandos de administración. Para poder inscribirse a los cursos, un alumno deberá primero iniciar una sesión en el sistema. Luego tendrá la posibilidad de seleccionar cursos y deshacer sus selecciones. Cuando el alumno cierre la sesión será inscripto en los cursos seleccionados. Por simplicidad, el sistema debe permitir una única sesión en curso.

2.3.1. Comando: `listar_cursos`

Formato: `listar_cursos`

Descripción: Muestra una lista de los cursos disponibles.

Salida: Un curso por línea, con el siguiente formato:

```
<idc>: <descripcion> (<materia>) Vacantes: <vacantes> ↩
    ↪Inscriptos: <cantidad>
```

Ejemplo:

```
listar_cursos
a1rw: Algoritmos I - Rosita W. (75.40) Vacantes: 60 ↩
    ↪Inscriptos: 0
a2rw: Algoritmos II - Rosita W. (75.41) Vacantes: 60 ↩
    ↪Inscriptos: 0
```

2.3.2. Comando: `listar_cursos_materia`

Formato: `listar_cursos_materia materia`

Descripción: Idem `listar_cursos`, pero filtrando el listado por materia.

Parámetros:

`materia`**:** Código de la materia.

Salida: Idem `listar_cursos`

Ejemplo:

```
listar_cursos_materia 75.41
a2rw: Algoritmos II - Rosita W. (75.41) Vacantes: 60 ↩
    ↪Inscriptos: 0
```

2.3.3. Comando: `sesion_iniciar`

Formato: `sesion_iniciar padron`

Descripción: Inicia una nueva sesión asociada a un padrón. Solo es posible ejecutar este comando si no hay otra sesión en curso.

Parámetros:

`padron`: Número de padrón del alumno.

Salida: `OK` en caso de no producirse error. Mensaje de error apropiado en caso contrario (ver sección 2.4).

Ejemplo:

```
sesion_iniciar 55555
OK
```

2.3.4. Comando: `sesion_inscribir`

Formato: `sesion_inscribir idc`

Descripción: Selecciona un curso para inscribirse al finalizar la sesión. Debe haber una sesión en curso para poder utilizar este comando. La inscripción al curso seleccionado se realizará cuando se cierre la sesión con `sesion_aplicar`.

Parámetros:

`idc`: Identificador del curso.

Salida: Mensaje apropiado en caso de error (ver sección 2.4). En caso de no producirse error debe informarse si el alumno quedó en condición regular o condicional:

```
OK: el padron <padron> sera inscripto como ↩
    ↪<regular|condicional>
```

Ejemplo:

```
sesion_inscribir a2rw
OK: el padron 55555 sera inscripto como regular
```

2.3.5. Comando: `sesion_ver`

Formato: `sesion_ver`

Descripción: Muestra el estado actual de la sesión. Muestra los cursos que fueron seleccionados con `sesion_inscribir`. [1]

Salida: Mensaje apropiado en caso de error (ver sección 2.4). En caso contrario, mostrar el padrón asociado a la sesión en la primera linea:

```
Padron: <padron>
```

Luego, si no hay cursos seleccionados:

```
No hay inscripciones
```

En caso contrario, mostrar un curso por línea:

```
<cid> (<regular|condicional>)
```

Ejemplo:

```
sesion_ver
Padron: 55555
a2rw (regular)
otro_curso (condicional)
```

2.3.6. Comando: `sesion_deshacer`

Formato: `sesion_deshacer`

Descripción: Des-selecciona el último curso que haya sido seleccionado con `sesion_inscribir`. Puede usarse repetidas veces siempre que quede algún curso seleccionado.

Salida: `OK` en caso de no producirse error. Mensaje de error apropiado en caso contrario (ver sección 2.4).

Ejemplo:

```
sesion_deshacer
OK
```

[1] Es decir, este comando no muestra los cursos en los que el alumno está actualmente inscripto, sino que solo muestra los cursos seleccionados en la sesión.

2.3.7. Comando: `sesion_aplicar`

Formato: `sesion_aplicar`

Descripción: Inscribe al alumno en los cursos seleccionados, y cierra la sesión.

Salida: `OK` en caso de no producirse error. Mensaje de error apropiado en caso contrario (ver sección 2.4).

Ejemplo:

```
sesion_aplicar
OK
```

2.4. Mensajes de error

El sistema debe imprimir los siguientes mensajes de error, en cada uno de los casos de error descriptos a continuación:

Id repetido:

```
Error: el curso con id "<idc>" ya existe
```

Curso no existente:

```
Error: el curso con id "<idc>" no existe
```

Alumno ya inscripto en un curso:

```
Error: el padron <padron> ya esta inscripto en el curso ↩
    ↪"idc"
```

Alumno no inscripto en un curso:

```
Error: el padron <padron> no esta inscripto en el curso ↩
    ↪"idc"
```

No hay una sesión en curso:

```
Error: no hay una sesion en curso
```

Ya hay una sesión en curso:

```
Error: ya hay una sesion en curso
```

El curso ya fue seleccionado en la sesión:

```
Error: la sesion ya contiene una inscripcion al curso ↩
    ↪"<idc>"
```

No hay acciones para deshacer:

```
Error: no hay acciones para deshacer
```

3. Pruebas

Junto con la especificación se provee de **pruebas automáticas para el programa completo**. Estas pruebas serán de utilidad para revisar que el programa cumpla con la especificación del protocolo de entrada/salida.

Para la aprobación del Trabajo Práctico es requisito que el programa implementado pase todas las pruebas.

3.1. Ejecución de las pruebas

Una vez descomprimido el archivo zip con las pruebas[1], se debe efectuar los siguientes pasos para correr las pruebas:

1. Compilar el programa (supongamos que se llama `tp1`)
2. Ejecutar[2]:

```
$ bash pruebas/correr-pruebas.sh ./tp1
```

Cada una de las pruebas está especificada en un archivo con extensión `.test` dentro de la carpeta de pruebas. Por ejemplo:

```
pruebas/
 \- correr-pruebas.sh
 \- prueba1.test
 \- prueba2.test
```

El script `correr-pruebas.sh` ejecutará el programa una vez por cada prueba, pasándole en la entrada estándar los comandos especificados en la prueba. Luego verificará que la salida estándar del programa sea exactamente igual a la esperada según el protocolo.

[1] Puede descomprimirse en cualquier lugar; para el ejemplo suponemos que se guardó en la misma carpeta que el TP. Es decir, el ejecutable `tp1` quedaría al mismo nivel que la carpeta `pruebas` (que contiene el archivo `correr-pruebas.py`).

[2] Para correr las pruebas es necesario disponer de un entorno con línea de comandos Bash y herramientas GNU. En Linux seguramente no sea necesario instalar nada. Existen varias implementaciones para Windows; por ejemplo MSYS o Cygwin. Para Mac OSX existe el paquete `coreutils`

4. Criterios de aprobación

A continuación describimos criterios y lineamientos que deben respetarse en el desarrollo del trabajo.

4.1. Utilización de estructuras de datos

Para la realización de este trabajo es necesario utilizar las estructuras de datos vistas en clase (vector dinámico, pila, cola, lista), además de crear las estructuras adicionales que se consideren necesarias.

Todas las estructuras deben estar implementadas de la forma más genérica posible y correctamente documentadas.

4.2. Programa

El programa debe cumplir los siguientes requerimientos:

- Debe estar adecuadamente estructurado y modularizado, utilizando funciones definidas de la forma más genérica posible, sin caer en lo trivial.
- El código debe ser claro y legible.
- El código debe estar comentado y las funciones definidas, adecuadamente documentadas.
- El programa debe compilar sin advertencias ni mensajes de error[3], debe correr sin pérdidas de memoria, uso de valores sin inicializar, o errores en general. Es decir que, el programa debe correr en valgrind sin errores.
- Además, claro, debe satisfacer la especificación de la consigna y pasar todas las pruebas automáticas.

4.3. Informe

El informe deberá consistir de las siguientes partes:

- **Carátula** con la información del alumno/a y el ayudante asignado (en caso de saberlo de antemano).
- **Análisis y diseño**: Describir la solución elegida para resolver cada problema propuesto, y cómo se lleva a cabo. En particular, mencionar cómo es el flujo del programa, qué algoritmos y estructuras de datos se utilizan, y cuál es el orden de ejecución en tiempo y espacio de cada operación.

[3] `-Wall -pedantic -std=c99`

- **Implementación**: Incluir aquí *todo* el código fuente utilizado (en formato `monoespaciado`, para facilitar su lectura).
- También *opcionalmente,* toda explicación adicional que consideren necesaria, referencias utilizadas, dificultades encontradas, cambios o mejoras que se podrían hacer a futuro y conclusiones.

El informe debe estar lo más completo posible, con presentación y formato adecuados. Por ejemplo, este enunciado cumple con los requerimientos de un informe bien presentado.

5. Entrega

El trabajo consiste en:

- El informe impreso.
- El informe digital, en formato `.pdf`
- Una versión digital de **todos** los archivos de código fuente, separados del informe, en un archivo comprimido (`.zip` o `.tar.gz`).

Los dos últimos deben enviarse a la dirección `tps.7541rw@gmail.com`, colocando como asunto:

```
TP1 - Padrón - Apellido
```

Se aclara que por código fuente se entiende todos los archivos `.h` y `.c`, el archivo `Makefile` para poder compilar, y todos los archivos adicionales que sean necesarios para ejecutar el programa. No deben entregarse nunca archivos `.o` u otros archivos compilados.

El informe impreso debe entregarse en clase. El plazo de entrega vence el **lunes 30 de septiembre de 2013**.