Arquitectura de sistemas

Abelardo Pardo

University of Sydney
School of Electrical and Information Engineering
NSW, 2006, Australia
Autor principal del curso de 2009 a 2012

Iria Estévez Ayres

Damaris Fuentes Lorenzo

Pablo Basanta Val

Pedro J. Muñoz Merino

Hugo A. Parada

Derick Leony

Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática
Avenida Universidad 30, E28911 Leganés (Madrid), España

Proyecto: Gestión de puntos del GPS.

Se recibe en la empresa SAUCEM S.L. un encargo de un cliente que quiere una aplicación de usuario para ser ejecutada en un dispositivo con capacidad para obtener un conjunto de puntos de un GPS (sistema de geoposicionamiento). El usuario puede manipular una colección de puntos mediante las operaciones típicas de altas, bajas y modificaciones, y además facilitarle la intercomunicación entre dispositivos para intercambiar estas colecciones.

La funcionalidad de la aplicación nos la manda el cliente descrita mediante una especificación de requisitos (conocida a veces coloquialmente como la "espec").

Requisitos del cliente

1. La aplicación se arranca desde un terminal de comandos y se manipula enteramente con el teclado.

2. Tras arrancar, la aplicación muestra un menú principal de texto con las posibles operaciones, cada una de ellas identificada con un número o letra diferente. Si cuando se está esperando una orden del usuario se pulsa la combinación de teclas `CTRL-D`, la aplicación debe terminar, no sin antes preguntar al usuario si desea guardar los puntos que hubiera en memoria (si los hubiese) en un fichero. Esta pregunta también debe realizarla al salir de manera normal del programa.

3. El usuario selecciona una operación y esta puede requerir que se introduzcan datos adicionales por el teclado. Al acabar una operación se vuelve a mostrar el menú y se espera un nuevo comando del usuario.

4. Las operaciones que ***debe incluir*** el menú principal son:

    1. Obtener un punto del GPS. Se almacena en las estructuras de datos internas de la aplicación y se le asigna un número. La aplicación debe mantener los puntos en el orden en el que se van obteniendo.

    2. Mostrar por pantalla los puntos obtenidos hasta el momento. Deben aparecer con su número asociado, las coordenadas, y un texto descriptivo (si ha sido previamente asociado al punto). Además, deberán estar en el siguiente formato:

```
ID      Latitude    Longitude     Altitude   Text
1          24.56        45.67        43.21   Calle Preciados, Madrid
2         223.54        34.65        30.23
3          56.87        54.02       224.89   Puerta del Sol, Madrid
4          40.33        -3.765      520.00   Auditorio Padre Soler, Univers...
```

    3. Dado el número de un punto, que se obtiene con la funcionalidad b, sobreescribir sus coordenadas con las actuales del GPS. La descripción se deja como está.

    4. Borrar un punto dado su número en la colección que muestra la funcionalidad b.

    5. Pedir el nombre de un fichero y guardar la colección de puntos en ese fichero.

    6. Pedir el nombre de un fichero y añadir los puntos que contenga a la colección de puntos actual.

    7. Reordenar los puntos en la colección por valor creciente de latitud o longitud según escoja el usuario.

    8. El usuario introduce una distancia en metros (número decimal) y la aplicación imprime los puntos en la colección que están a menos de esa distancia de la posición actual.

9. ***NUEVA (Para la entrega final):*** Dado el número de un punto, leer una línea de texto del teclado y asignarla a dicho punto. Modifica las funciones anteriores necesarias de acuerdo con este nuevo campo. En el caso de que la cadena tenga más de 30 caracteres, se ha de guardar completa en la colección de puntos, pero a la hora de mostrarla sólo se mostrarán los 30 primeros y 3 puntos suspensivos para indicar que hay más caracteres.

5. Al seleccionar una operación, esta puede requerir que se introduzcan datos adicionales por el teclado. Al acabar la operación se vuelve a mostrar el menú principal a la espera un nuevo comando del usuario. Si el usuario desea volver al menú principal antes de acabar la operación, pulsará Crtl-D.

6. La aplicación debe ser robusta, es decir, debe poder recuperarse sin problemas si en algún punto el usuario no introduce los datos adecuados (letras cuando se espera un número, una cadena vacía, etc.)

7. La aplicación debe hacer una gestión correcta de la memoria dinámica, es decir, sin fugas, accesos a porciones sin inicializar, liberaciones incorrectas, etc.

¡Aviso!

Tal y como suele suceder más a menudo de lo deseable en el contexto industrial, un cliente puede hacer ajustes a esta especificación ***mientras se está desarrollando el proyecto***. Por tanto, si tal circunstancia se produce, el equipo debe tratarla con normalidad.

Requisitos de la empresa

Además de los requisitos del cliente que tienen que ver con el producto terminado, SAUCEM S.L. tiene su propia política de desarrollo de productos orientada a mantener su imagen corporativa de empresa fiable, y a conseguir un ciclo de desarrollo eficiente. Estos requisitos son:

1. La solución del proyecto debe utilizar algún tipo de estructura dinámica (lista, árbol, hash, o similar).

2. El código debe estar dividido en ficheros de tal forma que se agrupen funciones similares. Los nombres de las funciones deben ser elegidos para facilitar su entendimiento y localizar rápidamente la funcionalidad buscada.

3. ***Todos los ficheros de código*** (`*.c`) ***y de definiciones*** (`*.h`) deben tener la estructura de las plantillas `CFile.templ` y `CHeaderFile.templ` respectivamente que encontraréis en el directorio `Plantillas`. En la cabecera se debe incluir una descripción detallada de las funciones o definiciones que contiene el fichero. Todos aquellos campos de la plantilla que no tengan contenido deberán borrarse. Por ejemplo, si tu código no contiene "macros", deberás borrar esta parte de la plantilla.

   Además, aquellas partes del código más delicadas, deben ir precedidas de un bloque de comentario que lo explique. No incluyas comentarios línea a línea, sino en bloques al comienzo de un conjunto de pasos.

4. Todas las funciones, variables globales, `#define`, definiciones de estructuras y definiciones de enumeraciones debe estar documentadas precediendo su definición con texto de las plantillas en los ficheros con nombres `Function.templ`, `Variable.templ`, `Macro.templ`, `Struct.templ` y `Enum.templ` respectivamente. En el caso de las funciones se debe explicar el papel de cada parámetro así como el resultado que devuelve. Los fragmentos a incluir con sus campos por completar los puedes encontrar todos en la carpeta `Plantillas` de tu directorio de trabajo.

5. Si el símbolo `DEBUG` está definido al compilar, la aplicación muestra mensajes de depuración por pantalla con las principales operaciones que realiza.

6. El código debe compilar sin ningún tipo de error ni advertencia cuando se utiliza la opción `–Wall`

del compilador **gcc**.

7. La aplicación debe hacer una gestión correcta de la memoria dinámica, es decir, sin fugas, accesos a porciones sin inicializar, liberaciones incorrectas, etc.

8. En la carpeta `Actas` del espacio de trabajo del grupo debe haber un acta para cada una de las reuniones de grupo que se han realizado. En ella debe constar el orden del día, los asistentes, quién ha realizado de moderador, quién ha escrito el acta y las decisiones que se han tomado.

9. El código debe estar creado y subido a Subversion de manera equilibrada ***por todos los miembros del equipo***. El comando **svn commit** registra el usuario que ha subido la nueva versión. Además, el comando **svn blame fichero.c** muestra quién ha sido el autor de cada una de las líneas de código del fichero. Se exige una contribución equilibrada de todos los miembros del equipo. Cuidado con hacer que un único miembro sea el que sube el código o los cambios a Subversion, como cada miembro tiene que contribuir, debe haber entradas de todos los miembros.

Hitos del proyecto

El proyecto se divide en cuatro hitos que se deben conseguir en ese orden. La planificación del curso establece unas fechas tentativas para la entrega de estos hitos, pero el que un proyecto se entregue sin haber cumplido los hitos no significa que esté suspenso, sino que la nota no será perfecta. Análogamente, si un proyecto completa los hitos antes de los estipulado, puede proseguir con los siguientes.

- Hito 1. Implementación del bucle de lectura de comandos y de las funcionalidades a) a c). Se obtiene la primera versión de la aplicación en la que aparecen todos los comandos que solicita el cliente. Cuando se ejecuta alguno de los que no está implementado se imprime un mensaje notificandolo.

- Hito 2. Implementación de las funcionalidades d) a f) de las requeridas por el cliente.

- Hito 3. Implementación de las funcionalidades g) a h) de las requeridas por el cliente.

- Hito 4. Implementación de la funcionalidad i).

Entrega de la versión parcial y final

A mitad de proyecto hay que entregar para su evaluación una versión parcial que incluya las funcionalidades a) a f) ejecutando correctamente (sin "segmentation fault" ni anomalías de memoria). En el caso de la escritura a fichero, no se requiere que la aplicación sea capaz de añadir datos a un fichero; sólo será necesaria la capacidad de la aplicación para sobreescribir un fichero. Tampoco se requiere que, a la hora de salir del programa, la aplicación pregunte al usuario si quiere guardar los datos. La versión final sí deberá cumplir estos requisitos.

La entrega de la versión parcial y final se hará a través de los directorios `Version_parcial` y `Version_final` respectivamente del espacio de trabajo del equipo en Subversion. En estos directorios, y antes del final de la sesión en la que se exige la entrega, debes copiar todos aquellos ficheros de código que se necesitan para compilar la aplicación (copiar los ficheros y ***añadirlos al controlador de versiones con el comando add***) y subirlos al servidor con el comando **commit**.

La corrección del proyecto se hará primero compilando el programa con el siguiente comando:

```
gcc -Wall -g -DDEBUG -I../lib ../lib/gps_as.c *.c -lm
```

Las operaciones con Subversion sobre estos directorios se desactivarán mediante el sistema de permisos una vez haya concluido el plazo de entrega. A partir de ese momento podrás hacer cambios en tu copia particular, pero el servidor de Subversion no admitirá que envíes esos cambios.

Cómo compilar el proyecto

Cómo compilar el proyecto

El proyecto necesita código auxiliar que está almacenado en la carpeta `lib` de tu espacio de trabajo compartido. Para compilar el proyecto debes añadir a tu comando de compilación las siguientes opciones:

```
gcc -Wall -g
    -I../lib [TUS OPCIONES]
    ../lib/gps_as.c [TUS FICHEROS]
    -lm
```

El comando anterior asume que estás compilando tu proyecto en una carpeta al mismo nivel que `lib`. Si decides trabajar en otra carpeta, asegurate de que la ruta a la carpeta `lib` es la correcta (puede que la ruta `../lib` ya no apunte a esa carpeta).

Una vez terminado, el proyecto debe tener 2 ejecutables: ejecutable normal para plataforma Intel y ejecutable con la opción de depuración para la plataforma Intel (ver requisito de la empresa 5). Para la fase de desarrollo se recomienda utilizar la versión para la plataforma Intel con la opción de depuración (es decir, compilar con la opción "**-DDEBUG**").

Para evitar teclear varias veces estas líneas tan elaboradas se recomienda escribir los cuatro comandos de compilación en cuatro ficheros de texto (el comando tiene que estar íntegro en una única línea) y ejecutarlos desde el intérprete con el comando "**source**" seguido del nombre del fichero con el comando a ejecutar.

Evaluación del proyecto

La siguiente tabla muestra la guía que se utiliza para evaluar las entregas del proyecto. Aquellas categorías que ocupan dos casillas, se evalúan con la de la nota más alta.

Aspecto	Excelente (100%)	Aceptable (75%)	Discreto (50%)	Insuficiente (0%)
1 Compilación del código con la opción –Wall	El código compila sin ningún tipo de advertencia o error.		El código compila con dos avisos.	El código no compila, o compila con más de dos avisos.
2 Ejecución del programa	El programa ejecuta con normalidad y acorde a la especificación.	El programa termina abruptamente en una ocasión o no cumple con la especificación en uno o dos aspectos.	El programa falla en dos pruebas o no cumple con la especificación en más de dos aspectos.	El programa falla con frecuencia o el programa no cumple con la especificación.
3 Gestión de memoria	La aplicación ejecuta sin ningún tipo de anomalía cuando se analiza con **Valgrind**.	Se detecta o una fuga de memoria, o una liberación de memoria incorrecta.	**Valgrind** detecta dos anomalías al ejecutar la aplicación.	Se detectan más de dos anomalías al ejecutar con **Valgrind**.
	Los principales pasos de la		La aplicación imprime	El programa no

4 Mensajes de depuración	aplicación imprimen mensajes de depuración cuando se define el símbolo `DEBUG`.		La aplicación imprime pocos mensajes de depuración.	imprime ningún mensaje de depuración
5 Código en múltiples ficheros, definiciones y prototipos en `.h`	El código está dividido en varios ficheros y las definiciones y prototipos incluidos en uno o varios ficheros con extensión `.h`.		El código está en un único fichero, o el fichero con las definiciones y prototipos está incompleto (o incorrecto).	El código está dividido de forma arbitraria, y el fichero `.h` contiene información incorrecta y no se incluye de forma adecuada.
6 Uso de las plantillas de ficheros y documentación en la cabecera.	Todos los ficheros siguen la plantilla, tienen todos los campos obligatorios rellenos y la documentación en la cabecera es intuitiva.		Uno o dos ficheros no siguen la plantilla, no tienen todos los campos de documentación rellenos, o la documentación en la cabecera no es suficiente.	Mas de dos ficheros no siguen la plantilla, tienen algunos campos sin rellenar, o la documentación en la cabecera no es suficiente.
7 Documentación de variables globales, estructuras y funciones	Todas las variables globales, estructuras y funciones están perfectamente documentadas.	En un fichero hay una variable, una función o una definición de estructura sin comentar.	Hay dos ficheros con variables, funciones o declaraciones de estructuras sin documentar.	Hay más de dos ficheros con carencias en la documentación de funciones, declaración de estructuras o variables globales.
8 Estilo de codificación	Todos los ficheros cumplen con todos los requisitos de la guía de estilo.		Algunos ficheros no cumplen con los requisitos de la guía de estilo. O algún criterio de la guía de estilo es ignorado completamente.	La mayoría de los ficheros no respetan la guía de estilo. O la mayor parte de criterios de estilo se ignoran.
9 La aplicación es intuitiva	El usuario entiende sin problemas las funciones, los datos a introducir y cómo introducirlos.		Los mensajes en general están bien, pero en alguna ocasión no se entiende bien qué hay que hacer.	El usuario se atasca frecuentemente por no saber cómo ejecutar la aplicación.
10 Actas de las reuniones	El equipo se ha reunido con frecuencia (al menos una vez a la semana) y las actas detallan su contenido.		El equipo no se ha reunido con la frecuencia deseada y/o las actas no reflejan la actividad real.	Las actas tienen muchas carencias, o hay un número muy bajo de ellas.
11 Trabajo				

El trabajo distribuido igualmente entre los miembros del equipo (usa `svn log`)	Todos los miembros del equipo por igual han subido ficheros al depósito y contribuido a su escritura.	Sólo una parte del equipo ha subido ficheros al depósito y contribuido a su escritura.	Solo una persona del equipo ha subido ficheros al depósito y contribuido a su escritura.

Los criterios están ordenados en orden de importancia. Tener correctos los dos primeros es condición esencial para aprobar el proyecto. Si el programa no compila o falla con frecuencia, no se considera acabado y la nota es un cero.

Funcionalidad extra

Para aquellos equipos que alcancen el Hito 4 antes de lo previsto ***y solo para aquellos***, y que además obtengan el visto bueno de su tutor, se puede extender la aplicación por uno o varios de los aspectos que te comentará el tutor llegado el momento (esta extensión no tendrá ningún tipo de efecto sobre la nota del proyecto).

Fuentes de "inspiración"

Cuando se emprende un proyecto de estas características es muy común el obtener información de múltiples fuentes. Cuando el proyecto empieza y los plazos de entrega están lejos, no hay problema de este tipo. Pero cuando nos vemos con el tiempo encima, la desesperación nubla la capacidad de decisión. Lo mejor para evitar problemas es tener muy claro en todo momento cuando esas fuentes son de uso razonable y cuando se pueden considerar una copia del trabajo. Te incluimos a continuación una serie de sugerencias.

- Comentar en el pasillo, en un aula, entre clase y clase cómo estamos resolviendo un aspecto especial de la práctica (no hay problema).
- Estoy estudiando con compañeros en la biblioteca y uno saca un papel para dibujar su problema en el proyecto y yo dibujo lo que hemos hecho en nuestro proyecto (no hay problema).
- Tras tener una de estas conversaciones, mi amigo me escribe y me dice que no le ha quedado claro. Le mando por correo una descripción textual más detallada de lo que hemos hecho (no hay problema).
- Recibo otro correo y todavía no lo ve claro. Corto y pego un trozo de nuestro código en el correo que le envío como respuesta ***(si, tenemos un problema)***. Fácil de detectar por la estructura del código. Por pequeño que sea el fragmento.
- Encuentro un trozo de código en Internet que hace lo que yo quiero, lo corto y pego, y tras unos cambios, ya funciona (no hay problema). Asegurate de incluir la referencia en el propio código como un comentario.
- (Basado en caso real) Trabajo en una empresa y le muestro el enunciado a mis colegas. A alguno de ellos le emociona y me manda parte de la solución ***(si, tenemos un problema)***. Fácil de detectar, cada persona escribe código de forma especial, es difícil que tu colega tenga el mismo nivel de C que tú. Además, seguramente ese colega presuma de haber resuelto esto y se puede llegar a comentar a los profesores (ha pasado).
- (Basado en caso real) Mi mejor amigo ha tenido la mala suerte de que le ha tocado en otro equipo que no avanza y está desesperado. Me pide que, por favor, le mande el código, pero me promete solemnemente que solo lo va a utilizar para leerlo y que no va a copiar nada. Es mi colega, somos como hermanos, no puedo decir que no. Se lo mando ***(sí, tenemos un problema)***. Tu colega incluirá algún fragmento en la entrega o lo que es peor, entregará uno de tus ficheros "por error" (ha sucedido).

- Trabajo en el proyecto en una academia. El profesor me explica algunos aspectos de la solución pero sin mostrar código (nosotros sin problemas, tú veras).
- En la academia se ilustra la solución del proyecto con código ejemplo ***(si, tenemos un problema)***. Recibiremos varias entregas casi idénticas, signo inequívoco de la resolución en cooperativa.

Todas estas situaciones se pueden resumir en dos consejos fundamentales. Tu código solo lo pueden ***ver*** los miembros de tu equipo, y tú no debes ver otro código que no sea el tuyo. Y cuando utilices fuentes de información, ***cítalas*** en la documentación que se incluye en la cabecera de cada fichero. En cualquier caso, ante la duda, ¡pregunta!.

En caso de que se detecte un caso de copia, como ya ha sucedido en ocasiones anteriores, se notificará a la dirección de la escuela para que abra las diligencias correspondientes y expediente a los alumnos involucrados. La nota del curso en tal caso será cero.

Cómo obtener un punto del GPS

La gestión del GPS se hace a través de una biblioteca que simplifica la interacción con el dispositivo y lo reduce a las dos siguientes operaciones.

- `int as_gps_read_location(double *latitude, double *longitude, double *altitude)`: Dados tres punteros a tres reales, la función almacena en las direcciones a las que apunta el valor actual obtenido del GPS. Cuidado, que si el GPS no ha obtenido las coordenadas, esta función almacena en los punteros tres valores cero. La función devuelve si el GPS tiene coordenadas válidas (un entero con valor a 1 para cierto, y 0 para falso).
- `int as_is_gps_ready()`: Devuelve cierto (entero con valor 1) si el GPS puede devolver coordenadas correctas y falso (entero con valor cero) en caso contrario.
