PRÁCTICA 1

Formato y fecha de entrega

La entrega se debe realizar en el apartado "Entrega y registro de EC" del aula de teoría, antes del día **23 de Marzo de 2014 a las 23:55**.

Se debe entregar un fichero en formato **ZIP** con el nombre *ApellidosNombre_PP_PRACT1.zip* que contenga:

1. Fichero PDF en el cual expliquéis los problemas encontrados en la realización de los ejercicios 1 a 5. Se debe adjuntar alguna captura de pantalla del entorno final pera cada ejercicio.
2. Carpeta/directorio **"UOCTranslator"** con el espacio de trabajo utilizado en el ejercicio 5. Antes de crear el fichero ZIP, eliminad los directorios temporales (desde el CodeLite podéis ejecutar el comando *clean*).

Nota: Es posible que al hacer la entrega el sistema cambie el nombre del fichero, no os preocupéis por este hecho.

Presentación

Durante las distintas prácticas desarrollaréis un único proyecto, del que iréis diseñando e implementando funcionalidades. En esta primera práctica instalaremos el entorno de programación y crearemos el esqueleto del proyecto, aprendiendo a definir librerías de código y configurar algunos de los parámetros de configuración del entorno de programación que permiten una mejor organización de los archivos de código.

Es muy importante que se realice esta práctica para asegurar que se tiene preparado su equipo para la realización de las siguientes prácticas, las cuales irán dirigidas a implementar el proyecto. Si tienes problemas en la instalación o configuración del entorno de programación, utiliza el foro del laboratorio para resolverlos.

Competencias

Transversales

- Capacidad de comunicación en lengua extranjera.

Específicas

- Capacidad de diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y re-utilización.
- Conocimientos básicos sobre el uso y la programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación a la ingeniería.

Objetivos

- Instalación y configuración de un entorno abierto de programación.
- Compilar un proyecto de código con múltiples archivos.
- Comprensión del proyecto a desarrollar.
- Familiarizarse con las funcionalidades y el uso del código proporcionado por los profesores.
- Recordatorio de la sintaxis, el estilo y el entorno de programación en C.

Recursos

Para realizar esta práctica dispones de los siguientes recursos:

Recursos Básicos

Laboratorio de Prácticas de Programación: Dispones de un laboratorio asociado a la asignatura. En este laboratorio puedes resolver todas las dudas sobre la programación en lenguaje C, y la instalación y utilización del entorno de programación.

Material de Laboratorio: En el laboratorio, aparte del colaborador docente, dispones de diferentes documentos que te ayudarán. En concreto, hay un *manual de C* y una *guía de programación en C*.

Transparencias de Síntesis: En el aula de teoría tienes las transparencias de síntesis. Para esta práctica te será de especial interés la TS2 y la TS3.

Vídeos CodeLite: En el aula de laboratorio encontraras una serie de vídeos sobre cómo utilizar el entorno de programación CodeLite.

Recursos Complementarios

Buscador Web: La forma más rápida de obtener información actualizada sobre los parámetros, funcionamiento y ejemplos de utilización de cualquier método estándar de C (y en general de cualquier lenguaje), es mediante un buscador web.

Criterios de valoración

Para la valoración de los ejercicios en que se pide contestar a una pregunta, se tendrá en cuenta:

- La adecuación de la respuesta a la pregunta formulada.
- Utilización correcta del lenguaje técnico y corrección en la redacción y ortografía.
- Claridad de la respuesta.
- Completud y nivel de detalle de la respuesta aportada.

En los ejercicios en que se pide que se entregue un código, primeramente el código entregado tendrá que compilar, en caso contrario no se puntuará el ejercicio, y en caso de que se faciliten juegos de prueba (públicos), que se pasen correctamente, si no se pasan, tampoco se puntuará el ejercicio. Si el código compila y pasa los juegos de prueba, se tendrá en cuenta los siguientes factores:

- El código pasa los juegos de prueba privados. Es decir, obtiene el resultado esperado dadas unas condiciones y datos de entrada diseñadas para probar algunas situaciones de funcionamiento normal y otros casos especiales. El formato de los juegos de prueba privados, será similar al de los públicos, entregados con el enunciado.
- El código entregado sigue la guía de estilo y las buenas prácticas de programación.
- Se separa correctamente la declaración e implementación de las acciones y funciones, utilizando los ficheros correctos.
- El código está correctamente comentado, valorando especialmente la utilización de comentarios en inglés.
- El grado de optimización en tiempo y recursos utilizados de la solución entregada.
- El código realizado es modular y estructurado, teniendo en cuenta la reutilización de código.

Descripción de la Práctica a realizar

La práctica consiste en implementar un traductor multilingüe, que permitirá traducir palabras a distintos idiomas. Durante las tres prácticas de la asignatura se irá completando diferentes partes del proyecto, de forma incremental.

Esta primera práctica contiene pequeños ejercicios con indicaciones que te permitirán aprender a crear proyectos, compilarlos y ejecutarlos. El objetivo principal es introducir algunos conceptos prácticos de programación y crear la estructura de ficheros y directorios que se utilizará durante el resto de prácticas. Lee atentamente las indicaciones dadas en cada ejercicio.

Nota: Los nombres de los tipos, los atributos y las operaciones deben escribirse en inglés. Los comentarios no es obligatorio hacerlos en inglés, aunque si que se valorará positivamente que se haga, ya que es el estándar. **Nota:** Hay que poner las declaraciones de los métodos en los ficheros de cabecera (*. h).

Ejercicio 1

Para la realización de las prácticas utilizaremos el entorno de programación CodeLite. Este entorno está disponible de forma gratuita para la gran mayoría de los sistemas operativos, incluyendo distintas versiones de Windows, Mac OS y Linux. Puedes encontrar toda la información sobre este entorno en su página web `www.codelite.org`. A continuación tenéis algunas indicaciones:

Windows: Descarga la versión más reciente del entorno de programación para Windows, teniendo en cuenta que necesitas una de las versiones que lleva integrado el MinGW. El archivo es un instalador, por lo tanto sólo hay ejecutarlo.

Mac OS: Descarga la versión más reciente del entorno de programación para Mac OS. Para poder compilar, debes tener instaladas las herramientas de desarrollo para OS. El archivo es un ZIP que contiene la aplicación y sólo hay que descomprimirlo y ya se puede ejecutar la aplicación. Si quieres puedes arrastrar la aplicación a la carpeta de aplicaciones.

Linux: El entorno se puede instalar directamente desde el gestor de aplicaciones de Linux en el caso de Ubuntu. Debes tener permisos de administrador y puede instalarse con la instrucción apt-get. Para ello, ejecuta los siguientes comandos en orden:

1. sudo apt-key adv –fetch-keys http://repos.codelite.org/CodeLite.asc
2. sudo apt-add-repository 'deb http://repos.codelite.org/ubuntu/ raring universe'
3. sudo apt-get update
4. sudo apt-get install codelite wxcrafter

Nota: A continuación tenéis el enlace directo a los archivos actuales para Windows y Mac OS, que deberían funcionar para la mayoría de gente:

Windows: `http://sourceforge.net/projects/codelite/files/Releases/codelite-5.2/codelite-5.2-mingw4.7.1.exe/download`

Mac OS: `http://sourceforge.net/projects/codelite/files/Releases/codelite-5.2/codelite_5.2_osx_10.8.app.zip/download`

Solución

E ste ejercicio es importante para asegurar que cuando empezéis a realizar las prácticas y otros ejercicios puntuales que puedan aparecer en alguna PEC, tengáis el entorno correctamente instalado en vuestro sistema. Si no habéis podido instalar correctamente el entorno de programación, enviad un correo a vuestro consultor de laboratorio para que os ayude a solucionar los problemas que hayáis tenido.

Ejercicio 2

Para asegurar que tienes el entorno de programación correctamente instalado y preparado para empezar a programar, crearemos un primer programa

Nota: Tienes a tu disposición el vídeo *CodeLite_NewProject* donde se muestran los pasos que seguiremos en este ejercicio.

Tarea 1

Arranca el entorno de programación y crea un nuevo proyecto **HelloWorld**, dentro de un entorno (*workspace*) con el mismo nombre.

Tarea 2

Crea un fichero **main.c** que contenga la función:

```
int main(int argc, char** argv);
```

la cual muestre el mensaje "hello world" por pantalla.

Tarea 3

Compila y ejecuta el programa para comprobar que funciona.

Solución

E n este ejercicio se muestra la creación de un proyecto de programación. Cuando se crea un nuevo proyecto en CodeLite ya contiene exactamente lo que se pide en este ejercicio, un fichero main.c con el método *main* que muestra un mensaje *hello world* por pantalla.

Para crear un nuevo proyecto, lo podemos hacer con la opción *New Project* del menú *Workspace*. Para este ejercicio, se tiene que especificar como nombre de proyecto *HelloWorld* y al aceptar se crea el entorno de trabajo y el proyecto con este nombre.

Para compilar el proyecto, podemos seleccionar la opción *build* del menú que aparece al presionar con el botón derecho sobre el proyecto o con la opción *Build Project* del menú *Build* (el proyecto creado debe estar como activo). También se puede hacer mediante el icono con un triangulo verde.

Finalmente, para ejecutar lo podemos hacer mediante la opción *Run* del menú *Build* o con el botón con la rueda dentada.

Hay teclas de acceso rápido para cada opción. Resulta muy útil aprender las teclas para tu sistema, ya que estas opciones se utilizan habitualmente.

Ejercicio 3

En Fundamentos de Programación has realizado pequeños programas que implementaban determinados algoritmos. Dada la simplicidad de los problemas tratados, todo el programa estaba contenido en un único archivo de código.

A medida que los programas crecen, surge la necesidad de dividir el código en diferentes archivos, que contienen partes de código relacionadas. Además, una práctica habitual es separar las declaraciones de los métodos y su implementación en ficheros diferentes. Las declaraciones se ponen en un archivo con extensión **.h*, mientras que su implementación se pone en ficheros con extensión **.c*.

En este ejercicio modificaremos el proyecto creado en el ejercicio anterior para distribuir el código en diferentes archivos. Tienes a su disposición los vídeos *CodeLite_ ConfigOutput* y *CodeLite_ CodeOrganitzation* donde se muestran los diferentes pasos que seguiremos en este ejercicio.

Tarea 1

Cambiar el código del ejercicio anterior para que el archivo de salida se genere en un directorio *bin* en la raíz del proyecto.

Tarea 2

Define un método

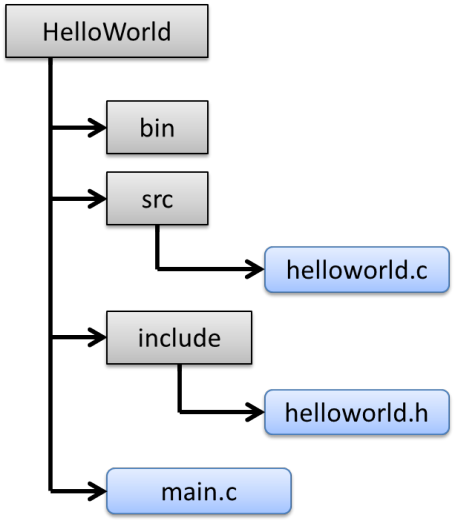
```
void sayHello();
```

que muestre el mensaje *"hello world"*. Se debe definir el método en un fichero **helloworld.h** dentro de una carpeta *include* en la raíz del proyecto. Implementa el método en un fichero **helloworld.c** dentro de un directorio *src*. Utiliza este método dentro del método *main*.

A continuación tenéis un esquema de cómo deben quedar los directorios y ficheros:

Solución

E n este ejercicio se ve cómo trabajar con proyectos con varios archivos y cómo configurar algunos aspectos básicos de la configuración de un proyecto. Para crear un nuevo archivo lo podemos hacer con la opción *Add new File* del menú que se muestra al pulsar el botón derecho sobre una de las carpetas del proyecto o el mismo proyecto. También se puede hacer mediante la opción *New File* del menú *File* .



Habrá que elegir uno de los dos tipos de archivos según si queremos crear un fichero de cabeceras (*.h) o un archivo de código (*.c) . En el vídeo se muestran los pasos a seguir. Como punto importante a tener en cuenta es que los archivos de código (*.c) que tenemos en el proyecto quedan incluidos cuando se compila, pero si tenemos archivos de cabecera (*.h) en directorios diferentes del directorio donde se encuentra el archivo que lo utiliza, habrá que añadir estos directorios adicionales a las opciones del compilador para este proyecto. Igualmente, si queremos cambiar el nombre o lugar donde se guardará el programa resultante, también hay que cambiar las opciones del proyecto .

Para acceder a las opciones del proyecto se hace con la opción *Settings* del menú que se abre pulsando el botón derecho sobre el proyecto. También se puede abrir con la opción *Open Active Project Settings* del menú *Workspace* (es necesario que el proyecto sea el proyecto activo). Dentro de las opciones, para cambiar el directorio o nombre de archivo de salida, hay que cambiar la opción *Output File* de la pestaña *Common Settings/General*. En esta misma pestaña podemos decidir el directorio desde el que se ejecutará la aplicación, con la opción *Working directory*. Para indicar los directorios en los que se buscarán los ficheros de cabecera (*.h), es necesario modificar la opción *Include Paths* de la pestaña *Common Settings/Compiler* .

Fijaros que podemos crear carpetas virtuales en el proyecto, que no tienen por qué coincidir con los directorios reales, pero que en caso de que coincidan, los nuevos ficheros se añaden por defecto en estos directorios.

Referente al código, el método *sayHello* se debe declarar en el fichero *helloworld.h*, lo que significa añadir la línea de código ***void*** *sayHello();*. La implementación del método la escribiremos en el fichero *helloworld.c*, que constará de un comando *printf* con el mensaje indicado. En el fichero *helloworld.c* deberemos añadir el fichero de cabeceras que hemos creado, con la línea *#include "helloworld.h"* al principio del fichero.

Ejercicio 4

Una forma muy habitual de reutilizar código es la utilización de librerías. Una librería es similar a una aplicación, con la diferencia de que no se puede ejecutar directamente (no existe ningún método *main*). Las librerías se utilizan desde aplicaciones, añadiendo funcionalidades. En Fundamentos de Programación ya has estado utilizando librerías, aunque no lo has hecho de forma explícita. Por ejemplo, cuando mostrábamos un mensaje por pantalla utilizando el comando *printf*, realmente estabas usando una librería, en la que está implementado este comando. En este ejercicio veremos cómo hacer nuestra propia librería y utilizarla desde dos aplicaciones diferentes.

Nota: Tienes a tu disposición el vídeo *CodeLite_Library* donde se ven los pasos a seguir para resolver las distintas tareas de este ejercicio.

Tarea 1

Crea un nuevo espacio de trabajo que tenga por nombre *HelloWorldApp*, en el que debes crear dos proyectos de tipo aplicación con nombres *HelloWorld1* y *HelloWorld2*.

Tarea 2

Configurar los dos proyectos anteriores para que generen su salida en un único directorio *bin* en la raíz del espacio de trabajo.

Tarea 3

Crear un nuevo proyecto **HelloWorldLib** de tipo *librería estática* dentro del espacio de trabajo anterior. Añadir a la librería los siguientes métodos:

```
void showHelloMessage();
void showAuthorName();
```

donde el método **showHelloMessage** mostrará el mensaje *Hello World* por pantalla, y el método *showAuthorName* mostrará tu nombre completo por pantalla. Es necesario que separes la declaración y la implementación en los archivos **helloWorld.h** y **helloWorld.c**, guardando los archivos de implementación (*. c) en un directorio *src* y los de cabecera (*. h) en un directorio *include*.

Tarea 4

Define un método:

```
void showAppName();
```

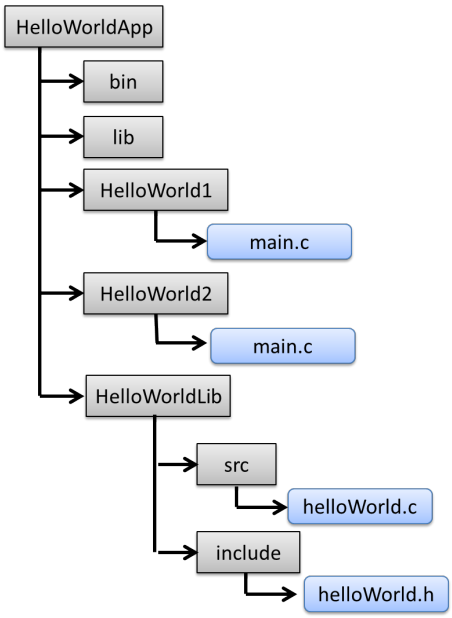
tanto en la aplicación *HelloWorld1* como en la aplicación *HelloWorld2* que muestre por pantalla el nombre de la aplicación, o sea, *HelloWorld1* o *HelloWorld2* según la aplicación.

Tarea 5

Agregar las llamadas a los métodos *main* de las dos aplicaciones, de modo que al ejecutarse la aplicación se muestra el nombre de la aplicación, el nombre del autor y el mensaje *Hello World*. A continuación tenéis un esquema de cómo deben quedar los directorios y ficheros:

Solución

E n este ejercicio se pide crear una librería. Una librería es un tipo de proyecto que no está pensado para ser ejecutado, sino que contiene métodos que son utilizados por otros proyectos. En una librería definiremos aquellos métodos (acciones o funciones) que nos pueden servir para varias aplicaciones. A la hora de programar no hay ninguna diferencia con lo que hacemos en el caso de las aplicaciones, los métodos se declaran en los ficheros de cabecera (*.h) y se implementan en los archivos de código (*.c).
En el vídeo se muestra todo el proceso necesario para resolver este ejercicio. Hay que tener en cuenta que si empezamos creando un espacio de trabajo nuevo, en vez de un proyecto nuevo, los proyectos que se añaden lo hacen por defecto en un directorio dentro del directorio del espacio de trabajo,



en lugar de compartir el mismo directorio como ocurría en los ejercicios anteriores. Esto afecta al utilizar rutas relativas para definir los directorios donde guardamos la salida.

Para resolver este ejercicio había que crear un espacio de trabajo con la opción *New Workspace* del menú *Workspace*, y darle por nombre *HelloWorldLib*. Posteriormente se tenía que crear dos aplicaciones *HelloWorld1* y *HelloWorld2* de la misma manera que hemos hecho en los ejercicios anteriores, configurando la salida en un directorio *bin* dentro de la carpeta del espacio de trabajo. Al igual que en el caso anterior, se tenía que separar las declaraciones de la implementación del único método que contienen estos proyectos. También se tenía que configurar la lista de directorios donde ir a buscar las cabeceras.

Además se pedía crear un nuevo proyecto de tipo librería, seleccionando el tipo *static library* como tipo de proyecto, y definiendo el método *showHelloWorld*, que será un método compartido por las dos aplicaciones. Al igual que en el caso de las aplicaciones, debíamos configurar los directorios donde ir a buscar los archivos de cabecera y el directorio de salida, que en este caso debía ser el directorio *lib*. Un punto importante es el hecho de que dependiendo de si trabajamos en Windows o en Linux/Mac, las librerías deben tener una extensión *lib* o *a* respectivamente.

Una vez creada y compilada la librería, para hacer que las aplicaciones la utilicen, debíamos añadir el archivo de cabecera de la librería en el código de las aplicaciones, al igual que se ha hecho en el caso de los ficheros de cabecera de los propios proyectos, y dentro de las opciones del proyecto, añadir el nombre de la librería en la opción *Libraries* de la pestaña *Common Settings/Linker*. En esta misma pestaña, hay que indicar el directorio donde ir a buscar las librerías, que en este caso es el directorio indicado como salida de la librería. Finalmente, hay que modificar las opciones de los proyectos con las aplicaciones, para indicar que también deben ir a buscar los archivos de cabecera en el directorio de archivos de cabecera de la librería.

Para compilar y ejecutar una aplicación que utiliza una librería, primero hay que asegurarse de que la librería se ha creado. Por lo tanto, primero tenemos que compilar el proyecto que define la librería. Se puede automatizar este proceso indicando que un proyecto depende de otro, y que por tanto cuando lo compilamos primero se debe compilar el otro proyecto. Esto lo podemos hacer con la opción *Build Order* del menú que se muestra al pulsar el botón derecho sobre un proyecto, y seleccionar el proyecto del que depende (la librería) .

Ejercicio 5

En los ejercicios anteriores hemos estado practicando la creación de diferentes tipos de proyectos con el entorno CodeLite y aprendiendo a configurar las opciones básicas de estos proyectos. En este ejercicio crearemos el proyecto que utilizaremos para las siguientes prácticas.

La práctica que realizaremos será un traductor. Este proyecto estará dividido en tres partes diferenciadas:

Programa: Será la aplicación que utilizará un usuario para acceder a las funcionalidades de traducción.

Librería: Definiremos una librería que contendrá los métodos genéricos para gestionar los diccionarios y facilitar las funcionalidades de traducción.

En las siguientes tareas se irá indicando los diferentes pasos a seguir. Utiliza lo que has hecho en los ejercicios anteriores como ejemplo.

Tarea 1

Crea un nuevo espacio de trabajo con nombre **UOCTranslator**, que contenga una aplicación ejecutable con nombre **Translator**, y una librería con nombre **UOCTranslatorLib**. Configura el entorno para que la aplicación utilicen esta librería, y que los resultados se generen en los directorios **bin** y **lib**, igual que hemos hecho en el ejercicio anterior.

Tarea 2

Dentro de la librería diferenciaremos distintas partes. Para esta práctica, crearemos un fichero de cabeceras **dictionary.h**, donde pondremos los métodos relacionados con la creación y gestión de diccionarios. En las siguientes prácticas definiremos otras partes.

Crea el fichero de cabeceras *dictionary.h* dentro del directorio *include* de la librería. Añade las siguientes declaraciones a estos ficheros:

dictionary.h

```
// Calc the edition distance between two strings
int getDistance(char* ref,char* word);
// Get word information
int getWordInfo(char* str,char* word,int* langID,int* wordID);
```

El método *getDistance* nos devolverá el número de letras diferentes entre las dos palabras que se le pasen como parámetro. Se empieza a comparar las dos palabras por el principio, y a partir de la primera letra diferente, se cuentan las letras diferentes que hay hasta el final de la palabra más larga. Por ejemplo, entre "hola" y "hospital", hay una distancia de 6 ("spital") , al igual que entre "hospital" y "hola".

El método *getWordInfo* coge la información contenida en una cadena de caracteres *str* en el formato "word ; langId ; wordId" , en la que word es una cadena de caracteres correspondiente a una palabra, langId es un entero positivo que indica el código del idioma y wordId es un entero positivo correspondiente al identificador de esta palabra en este idioma. Es importante que entre la palabra y el punto y coma haya un espacio. Los valores se devuelven en los parámetros de salida con el mismo nombre.

Solución

E n esta tarea sólo se pide la creación del fichero de cabeceras, por lo que no se debe implementar ningún método, simplemente añadir su declaración.

Tarea 3

Busca información en Internet sobre el comando *sscanf*. Explica cómo la has buscado y la dirección de la página donde has encontrado la información. Explica brevemente para qué sirve y pon un ejemplo de código en que se vea cómo se utilizaría para leer una cadena de caracteres con el formato explicado en la tarea anterior para el comando *getWordInfo*.

Solución

E l comando *sscanf* nos permite extraer el contenido en una cadena de caracteres, siguiendo un formato predefinido. Funciona de la misma forma que el método *scanf* utilizado en Fundamentos de Programación para la lectura de un entero desde teclado, pero en este caso la entrada es una cadena de caracteres. Viene a ser el comando opuesto a *printf*. Para encontrar información sobre este comando simplemente era necesario escribir el nombre del comando en un buscador como Google. Si queremos acotar más la búsqueda, podemos añadir el lenguaje de programación, buscando "C sscanf" o si queremos ceñirnos a comandos estándares, buscar por "ANSI C sscanf". En cualquiera de los casos, los primeros resultados contienen la información que necesitábamos. Por ejemplo, en la última búsqueda, el primer resultado que encontramos es la página: http://c.conclase.net/librerias/?ansifun=sscanf

Nota: Los resultados pueden ser distintos según el buscador o el idioma en qué naveguemos.

A partir de esta página vemos que este comando requiere incluir la librería "*stdio.h*", y que siempre toma dos parámetros fijos, correspondientes a la cadena de caracteres que queremos analizar y a su formato. Además se debe añadir un parámetro para cada elemento que queramos extraer de la cadena de caracteres. El formato de la cadena de caracteres se especifica igual que en el caso de *printf*.

Para extraer la información en el caso de *getWordInfo*, deberíamos utilizar el siguiente código:

```
int getWordInfo(char* str,char* word,int* languageID,int* wordID) {
        sscanf(str,"%s ; %d ; %d",word,languageID,wordID);
        return 1;
}
```

Dado que sabemos que tanto el identificador de idioma como el de palabra tienen que ser enteros positivos, podemos utilizar el valor de retorno para indicar si estos son correctos o no, quedando el código de la siguiente forma:

```
int getWordInfo(char* str,char* word,int* languageID,int* wordID) {
        *languageID=-1;
        *wordID=-1;
        sscanf(str,"%s ; %d ; %d",word,languageID,wordID);
        return *languageID>0 && *wordID>0;
}
```

Tarea 4

Modifica el método *main* de la aplicación Translator con el siguiente código:

```
int main(int argc, char **argv)
{
        char word1[128];
        char word2[128];
        int langId1,wordId1;
        int langId2,wordId2;
        int d1,d2;

        getWordInfo("hospital ; 1 ; 1",word1,&langId1,&wordId1);
```

```
        getWordInfo("hola ; 1 ; 2", word2, &langId2, &wordId2);

        d1=getDistance(word1, word2);
        d2=getDistance(word2, word1);

        printf("[%d,%d] => %s\n", langId1, wordId1, word1);
        printf("[%d,%d] => %s\n", langId2, wordId2, word2);

        printf("d(%s,%s)=%d\n", word1, word2, d1);
        printf("d(%s,%s)=%d\n", word2, word1, d2);

        return 0;
    }
```

Indica cual es la salida del programa.

Solución

T eniendo en cuenta la descripción de los métodos, la salida esperada para el programa debería ser:

```
[1,1] => hospital
[1,2] => hola
d(hospital,hola)=6
d(hola,hospital)=6
```