Prácticas de Programación – Práctica 1

Formato y fecha de entrega

La entrega debe hacerse en el apartado “Entregas y registro de EC” del aula de teoría, antes del día **3 de abril de 2016 a las 23:55.**

Se debe entregar un fichero en formato ZIP con el nombre *CApellidosNombre_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 para cada ejercicio.
2. Carpeta / directorio “**UOCRaceManager**” 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 diferentes prácticas desarrollaremos un único proyecto, del que iréis diseñando e implementando funcionalidades. En esta primera práctica implementaremos la funcionalidad principal de la aplicación de gestión de carreras. Posteriormente se extenderá dicha funcionalidad para completar la aplicación.

Competencias

Transversales

- Capacidad de comunicación en lengua extranjera

Específicas

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

Objectivos

- Compilación de un proyecto de código y un proyecto de pruebas.
- Comprensión del proyecto a desarrollar.
- Familiarización 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 disponéis de los siguientes recursos:

Recursos Básicos

Laboratorio de Prácticas de Programación: Disponéis de un laboratorio asignado a la asignatura. En este laboratorio podéis 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, disponéis de diferentes documentos que os ayudarán. En concreto, hay un *manual de C* y una *guía de programación en C.*

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

Vídeos CodeLite: En el aula de laboratorio encontraréis una serie de videos 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 evaluación

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
- Completitud y nivel de detalle de la respuesta aportada.
- El código obtiene el resultado esperado dadas unas condiciones y datos de entrada diseñados para probar algunas situaciones de funcionamiento normal y otros casos especiales.
- 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 en la solución entregada.
- El código realizado es modular y estructurado, teniendo en cuenta la reutilización del código.

Descripción de la Práctica a realizar

La práctica consiste en implementar una aplicación que permite gestionar un calendario de carreras deportivas donde los diferentes participantes podrán consultar información, inscribirse en las carreras y consultar sus resultados.

Durante las tres prácticas de la asignatura se irá completando diferentes partes del proyecto de manera incremental.

El proyecto consiste en la implementación de dos interfaces. En la primera que diseñaremos, los administradores de la aplicación podrán crear carreras y consultar información básica sobre las mismas. En posteriores prácticas implementaremos la interfaz de usuario donde estos se podrán registrar a las carreras.

Nota: Los nombres de los tipos, de los atributos y de las operaciones se deben escribir en inglés. Es altamente recomendable redactar los comentarios también en inglés.

Nota: Hay que poner las declaraciones de las operaciones en los ficheros cabecera (*.h).

Ejercicio 1

El proyecto en esta primera práctica contiene tres partes claramente diferenciadas:

Aplicación RaceManager: Es la aplicación que ejecutarán los administradores de la aplicación y que servirá para registrar nuevas carreras y actualizar su información.

Librería RaceMgrLib: Contendrá los tipos de datos, la definición de las estructuras para la gestión de la información y operaciones comunes a las aplicaciones.

Librería TestLib: Contiene operaciones útiles para poder testear la funcionalidades programadas.

Tarea 1

Cread un nuevo entorno de trabajo UOCRaceManager con la misma estructura que la proporcionada en el enunciado. Tened en cuenta los 3 proyectos que debe contener la librería así como las 3 posibles vistas comentadas en la práctica 0: Debug, Release y Test.

Solución

En esta tarea únicamente se debe preparar el entorno para poder desarrollar las prácticas. En la solución simplemente se espera una breve descripción de los pasos que se han hecho o tan sólo una imagen de una captura de pantalla donde se vea el entorno preparado.

Tarea 2

En la librería *RaceMgrLib* existen los ficheros *helpers.c* i *helpers.h*. Estos ficheros contienen funciones auxiliares que nos permitirán realizar algunas operaciones comunes del programa.

Dentro de estas operaciones hay la función *helpers_clearScreen*() que contiene una llamada específica sólo para Sistemas Operativos Windows. El objetivo de esta función es el de limpiar la pantalla.

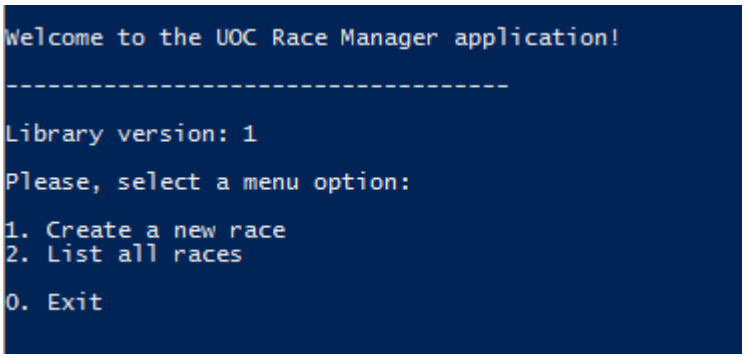
Adaptad esta función a vuestro Sistema Operativo y verificad que funciona correctamente. Encontraréis los comentarios de cómo adaptarla en el mismo código.

Solución

Los usuarios de Windows no necesitáis hacer nada en esta tarea. Los usuarios que tengáis Linux o iOS debéis comentar la línea específica de Windows y descomentar la línea específica de Linux / iOS.

Tarea 3

Probad de compilar y ejecutar el código proporcionado. Verificad que os aparece el siguiente menú:



Solución

Se puede adjuntar una captura de pantalla del programa inicial compilado y ejecutado que debería coincidir con la imagen del enunciado.

Ejercicio 2

Lo primero que hace la aplicación RaceManager es mostrar el menú inicial para poder actuar en consecuencia.

La interacción con el usuario consiste en un bucle en que se va procesando la entrada introducida por el usuario hasta que este pulsa la tecla ‘0’ para salir. Este bucle está implementado en la función *mgrMain_displayMenu* del fichero *mgrMain.c*

Tarea 1

Modificad la función *mgrMain_displayMenu* para que entre en un bucle que gestionará la interacción con el usuario. La opción escogida por el usuario viene devuelta en la variable *option*. La operación saldrá del bucle, con lo que retornará a la función principal, cuando el usuario pulse la tecla '0'

Implementad la funcionalidad del bucle teniendo en cuenta las 2 opciones disponibles. En caso que el usuario seleccione '1', se llamará a la función para crear nuevas carreras, mientras que si selecciona la opción '2', listará todas las carreras que están registradas en el sistema.

Solución

El código debe entrar en un bucle en el que se vayan gestionando las entradas del usuario hasta que la opción elegida sea igual al carácter '0'. En caso de que sea '1' debe llamar a la función mgrMain_newRace, mientras que en caso de que sea '2' debe llamar a la función mgrMain_listRaces.

```
void mgrMain_displayMenu()
{
	char option;

	option = mgrMenu_mainMenu();

	///
	/// EX.2.1: Manage the option introduced by the user
	///
	while (option != '0')
	{
		switch(option)
		{
			case '1':	// Create a new race
				mgrMain_newRace();
				break;

			case '2':	// List all races
				mgrMain_listRaces();
				break;

			default:
				break;
```

```
            }
            option = mgrMenu_mainMenu();
        }

        /// END OF EX.2.1
}
```

Ejercicio 3

Cuando se quiere crear una nueva carrera, el algoritmo llama en primer término a la función *mgrMenu_newRace* que pide los datos al usuario y los utiliza para rellenar la estructura que devolverá con toda la información. Posteriormente llama a la función *ops_createRace* que se encarga de validar que no hay ningún error en los datos introducidos y, en caso que todo sea correcto, introduce la información en la base de datos.

Tarea 1

La función *mgrMenu_newRace* gestiona los datos introducidos por el usuario. En el código proporcionado en la práctica, todos los datos introducidos son almacenados en variables locales de la función. Completad la función para que retorne una estructura con toda la información de la nueva carrera.

Nota: Encontraréis la definición de la estructura con los campos que utiliza en el fichero cabecera *races.h*

Nota: Llamad siempre a la función *races_initializeStruct* cuando creéis una nueva estructura del tipo *race*. Esta función simplemente inicializa todos los campos de la estructura a 0.

Solución

Hay una serie de aspectos a tener en cuenta en este ejercicio:

- Al retornar un puntero, lo primero de todo que debe hacerse es reservar memoria para almacenar su información.
- Recién hecho el malloc, deberemos llamar a la función races_initializeRaceStruct tal y como se indica en el enunciado.
- Los strings se copian utilizando la función strcpy.

- Finalmente, debemos asegurar que todos los campos de la estructura son rellenados con la información adecuada.

```
///
/// EX.3.1: Fill the fields of the struct with the proper information.
///

retValue = (tRace *)malloc(sizeof(tRace));
races_initializeRaceStruct(retValue);
strcpy(retValue->name, name);
strcpy(retValue->province, province);
strcpy(retValue->location, location);
retValue->distance = distance;
retValue->dateTime.year = dateTime.year;
retValue->dateTime.month = dateTime.month;
retValue->dateTime.day = dateTime.day;
retValue->dateTime.hour = dateTime.hour;
retValue->dateTime.minute = dateTime.minute;

/// END OF EX.3.1
```

Tarea 2

La segunda parte del algoritmo para crear una nueva carrera consiste en validar los campos introducidos por el usuario y, en caso que todo sea correcto, añadirla a la base de datos. Esta funcionalidad está gestionada por la función *races_addNewRace* del fichero *races.c* de la librería.

Modificad la función *races_addNewRace* para que valide los campos de la estructura y retorne un código de error en caso que haya cualquier error, teniendo en cuenta que:

1. El nombre de la carrera deberá tener como mínimo 3 caracteres. En caso contrario retorna INVALID_NAME.

2. La carrera únicamente se puede efectuar en una de las siguientes 4 provincias: Barcelona, Tarragona, Lleida o Girona. En caso contrario retorna INVALID_PROVINCE.
3. La población de la carrera deberá tener como mínimo 3 caracteres. En caso contrario retorna INVALID_LOCATION.
4. La distancia debe estar comprendida entre los valores indicados por las constantes MIN_RACE_DISTANCE y MAX_RACE_DISTANCE. En caso contrario retorna INVALID_DISTANCE.
5. La fecha y la hora deben ser datos válidos, teniendo en cuenta que el año debe estar comprendido entre los años marcados por las constantes MIN_RACE_YEAR y MAX_RACE_YEAR, y teniendo también en cuenta que el día debe ser un día válido acorde al mes indicado. En caso contrario retorna INVALID_DATETIME.

   Nota: No tendremos en cuenta los años bisiestos y entendemos que Febrero acepta siempre hasta 29 días.

Solución

Hay varias formas de solucionar este ejercicio, pero se aconseja dividir cada una de las validaciones en funciones específicas. También se debe tener en cuenta que, en caso de que la comprobación sea incorrecta, la función devolverá inmediatamente, y no se dará de alta la carrera en el sistema.

Aparte de las comprobaciones de la longitud de los nombres, para la comprobación de las provincias sería conveniente almacenar los valores aceptados en un array y comprobar que la provincia introducida está dentro del array. De esta manera, si más adelante se añaden más valores, estos nuevos valores sólo deberán ser añadidos al array, sin necesidad de modificar el código.

Por otra parte, el otro aspecto diferente era cómo gestionar los días válidos de cada mes. La forma más fácil es también almacenando el número de días válidos de cada mes en un array, y comparar el día introducido con el valor guardado en el índice del array.

```
///
/// EX.3.2: Check the race values are correct.
///

// Is the name valid?
error = races_checkName(race->name);
if (error != RACE_NOERR)
{
      return error;
}
```

```
// Is the province valid?
error = races_checkProvince(race->province);
if (error != RACE_NOERR)
{
      return error;
}

// Is the location valid?
error = races_checkLocation(race->location);
if (error != RACE_NOERR)
{
      return error;
}

// Is the distance correct?
error = races_checkDistance(race->distance);
if (error != RACE_NOERR)
{
      return error;
}

// Is the date and time correct?
error = races_checkDateTime(race->dateTime);
if (error != RACE_NOERR)
{
      return error;
}


/// END OF EX.3.2
```

```
/*
 * Function:    races_checkName
 * Description: Verifies the name is valid
 * Arguments:   name: The race name
 * Returns:     The error message
 */
raceError races_checkName(char * name)
{
      if (strlen(name) < 3)
      {
            // It can't be an empty name
            return INVALID_NAME;
      }



      return RACE_NOERR;
}



/*
 * Function:    races_checkProvince
 * Description: Verifies the province is valid
 * Arguments:   province: The province value
 * Returns:     The error message
 */
raceError races_checkProvince(char * province)
{
      char  races_validProvinces[NUM_OF_PROVINCES][MAX_LOCATION_LENGTH]
= {"Barcelona", "Tarragona", "Lleida", "Girona"};

      int i;
      for(i = 0; i < NUM_OF_PROVINCES; i++)
      {
```

```
            if (!strcmp(races_validProvinces[i], province))
            {
                  return RACE_NOERR;
            }
      }

      return INVALID_PROVINCE;
}


/*
 * Function:    races_checkLocation
 * Description: Verifies the location is valid
 * Arguments:   location: The location value
 * Returns:     The error message
 */
raceError races_checkLocation(char * location)
{
      if (strlen(location) < 3)
      {
            // It can't be an empty location
            return INVALID_LOCATION;
      }

      return RACE_NOERR;
}


/*
 * Function:    races_checkDistance
 * Description: Verifies the distance is valid
 * Arguments:   distance: The distance value
 * Returns:     The error message
 */
```

```
raceError races_checkDistance(int distance)
{
      if ((distance < MIN_RACE_DISTANCE) ||
          (distance > MAX_RACE_DISTANCE))
      {
            // Distance out of the boundary limits
            return INVALID_DISTANCE;
      }

      return RACE_NOERR;
}



/*
 * Function:    races_checkDateTime
 * Description: Verifies the dateTime is valid
 * Arguments:   dateTime: The dateTime value
 * Returns:     The error message
 */
raceError races_checkDateTime(tDateTime dateTime)
{
      int days[12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

      if ((dateTime.year < MIN_RACE_YEAR) ||
          (dateTime.year > MAX_RACE_YEAR) ||
          (dateTime.month < 1) || (dateTime.month > 12) ||
          (dateTime.day < 1) ||
          (dateTime.day > days[dateTime.month - 1]) ||
          (dateTime.hour < 0) || (dateTime.hour > 23) ||
          (dateTime.minute < 0) || (dateTime.minute > 59))
      {
            // Distance out of the boundary limits
            return INVALID_DATETIME;
      }
```

```
    return RACE_NOERR;
}
```

Ejercicio 4

En el menú inicial, la otra opción que podía escoger el usuario es listar todas las carreras disponibles, lo cual se realiza en la función *mgrMain_listRaces* del fichero *mgrMain.c*. De nuevo, este algoritmo consta de 2 partes. En una primera, el menú muestra todas las carreras disponibles en la base de datos con un índice a su lado. Posteriormente, el usuario podrá seleccionar uno de estos índices para mostrar información detallada sobre aquella carrera.

Tarea 1

Implementad la función *mgrMain_listRaces* para que muestre la lista de todas las carreras disponibles en la base de datos, lo cual se realizará llamando a la función *mgrMenu_listAll*.

Solución

De la misma manera que en el ejercicio 2, aquí tenemos un bucle donde el usuario introduce su opción mediante la llamada a la función mgrMenu_listAll

```
    ///
    /// EX.4: List all registered races.
    ///


    int option;
```

```
option = mgrMenu_listAll();

while (option != 0)
{
      option = mgrMenu_listAll();
}

/// END OF EX.4
```

Tarea 2

Gestionad la opción seleccionada por el usuario para que muestre por pantalla la información detallada de la carrera. Esta gestión se debe implementar en un bucle del que saldrá cuando el usuario presiones la tecla '0'.

La información detallada de la carrera se implementa llamando a la función *mgrMenu_information*.

En caso de introducir un índice inválido, la función debe mostrar el mensaje: "The value introduced is not a valid index", tal y como se muestra en la captura de pantalla.

```
List of registered races
------------------------
001 - Marathon of Barcelona
Select a race to display its information (0 to return to main menu)
2
The value introduced is not a valid index
Press any key to continue.
```

Nota: Podéis utilizar la función *races_getRace* per obtener el puntero a la carrera almacenada en el índice introducido por el usuario.

Solución

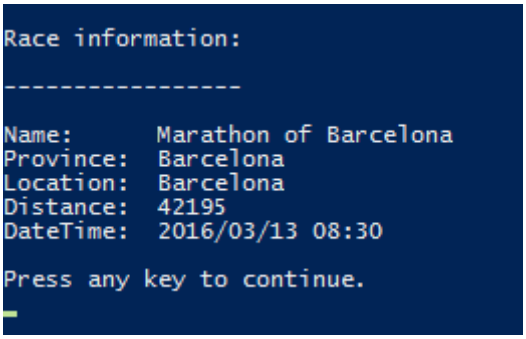
Se utiliza la función races_getRace para obtener el apuntador a la estructura que está guardada en el índice, teniendo en cuenta que este índice comienza en 0 y, por tanto, se debe restar 1 a la opción seleccionada. Hay que gestionar también el error en caso de que no haya ninguna carrera registrada en ese índice, o, de otra manera, mostrar su información mediante la llamada a mgrMenu_listAll

```
	///
	/// EX.4: List all registered races.
	///

	int option;
	tRace * race;

	option = mgrMenu_listAll();

	while (option != 0)
	{
		// Display the extended information of the selected race.
		race = races_getRace(option - 1);
		if (race == NULL)
		{
			printf("\nThe value introduced is not a valid index\n");
			helpers_pressAnyKey();
		}
		else
		{
			mgrMenu_information(race);
		}
		option = mgrMenu_listAll();
	}

	/// END OF EX.4
```

Tarea 3

Modificad la función *mgrMenu_information* para que muestre por pantalla la información completa de la carrera con el formato especificado en la siguiente captura de pantalla:

Nota: Tened en cuenta el formato de la fecha con 4 dígitos para el año y 2 dígitos para el resto de valores



Solución

Este es un ejercicio sobre presentación de datos por pantalla mediante el uso del comando printf. Es importante mostrar los datos con los dígitos y el formato que corresponde tal y como se comenta en el enunciado.

```
            ///
            /// EX.4.3: Display the race extended information
            ///

            // We found it, display its information
            printf("Name:      %s\n", race->name);
            printf("Province:  %s\n", race->province);
            printf("Location:  %s\n", race->location);
            printf("Distance:  %d\n", race->distance);
            printf("DateTime:     %04d/%02d/%02d   %02d:%02d\n",   race-
>dateTime.year,

            race->dateTime.month,
```

```
        race->dateTime.day,

        race->dateTime.hour,

        race->dateTime.minute);

        /// END OF EX.4.3
```

Ejercicio 5

Finalmente, una vez tenemos unas cuantas carreras introducidas en el sistema, en el momento de crear una nueva carrera debemos validar que su nombre no ha estado previamente introducido en la base de datos.

Tarea 1

Volved a la función *races_addNewRace* y, entre las comprobaciones previas que se hacían, validad que el nombre de la carrera no coincide con el nombre de ninguna de las otras carreras que ya han sido registradas en el sistema. En caso que el nombre exista, debéis retornar el error RACE_ALREADY_EXISTS.

Solución

Lo importante de este ejercicio es comprobar que habéis sido capaces de deducir que el primer puntero de la lista está almacenado en la variable global races_firstRace y, a partir de esta, puede ir recorriendo la lista mediante el apuntador a nextRace. Sabremos que tendremos recorrida toda la lista cuando el apuntador a nextRace sea NULL.

```
raceError races_checkName(char * name)
{
      tRace * currentRace;
```

```
	currentRace = races_firstRace;

	if (strlen(name) < 3)
	{
		// It can't be an empty name
		return INVALID_NAME;
	}

	if (!strcmp(name, currentRace->name))
	{
		return RACE_ALREADY_EXISTS;
	}

	while (currentRace->nextRace != NULL)
	{
		currentRace = currentRace->nextRace;
		if (!strcmp(name, currentRace->name))
		{
			return RACE_ALREADY_EXISTS;
		}
	}

	return RACE_NOERR;
}
```