PEC2

Formato y fecha de entrega

La PEC se debe entregar antes del día **07 de abril de 2017 a las 23:59**. Para la entrega se deberá entregar un fichero en formato **ZIP** que contenga:

- Este fichero con las respuestas a los distintos ejercicios que se plantean.
- Los ficheros de código que se piden, respetando el nombre indicado en el enunciado.

La entrega se hará en el apartado de entregas de EC del aula de teoría.

Presentación

En esta PEC empezaremos a trabajar tanto con las estructuras del lenguaje algorítmico como con pequeños programas. Se trabajará con los tipos de datos, la modularidad y la composición iterativa.

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 los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación a la ingeniería.

Objetivos

- Saber modularizar el código utilizando acciones y funciones
- Saber crear y utilizar archivos de cabecera.
- Saber definir tipos de datos estructurados.
- Saber definir y utilizar vectores.
- Saber utilizar la composición iterativa.
- Crear pequeños programas en C.
- Preparar pruebas adecuadas para los algoritmos/programas.

Recursos

Para realizar esta actividad tenéis a vuestra disposición los siguientes recursos:

Básicos

- Materiales en formato web de la asignatura de las 7 primeras semanas.
- Laboratorio de C.

Complementarios

- Internet. La forma más efectiva de encontrar información sobre cualquier duda sobre C es la búsqueda a través de un buscador.

Criterios de valoración

Cada ejercicio lleva asociada la puntuación sobre el total de la actividad. Se valorará tanto la validez de las respuestas como su completitud.

- En ejercicios en que se pide lenguaje algorítmico, se debe respetar el formato. Recuerda consultar el documento *Nomenclator* que resume la notación algorítmica que utilizamos en la asignatura (lo encontraréis en la Wiki).
- En el caso de ejercicios en lenguaje C, estos tienen que compilar para ser evaluados. En tal caso, se valorará:
    - que funcionen
    - que se respeten los criterios de estilo: revisad la *Guía de estilo de programación en C* que tenéis en la Wiki.
    - que el código esté comentado
    - que las estructuras utilizadas sean las correctas

[25%] Ejercicio 1: Tipos de datos estructurados

Resuelve tanto en **lenguaje algorítmico** como en **lenguaje C.**

Define los tipos estructurados que permitan representar los siguientes objetos, y una variable de cada uno de los tipos creados.

Nota: El tipo de datos utilizado para cada campo de las estructuras debe adecuarse a la información que guarda. Es posible que algunos de ellos requieren definir otros tipos de datos adicionales, ya sean enumerados u otras estructuras.

a) Un tipo **tStudent** para guardar la información de un estudiante de primaria que solicita acceso a un colegio de secundaria.

Se necesita almacenar:

- nombre del estudiante
- apellidos del estudiante
- sexo
  - boy
  - girl
- si es o no celiaco
- si tiene o no hermanos en el centro para el que solicita el acceso
- código del centro
  - rango: 1 - 99,999
- fecha en la que el estudiante entrega la inscripción
  - dia
  - mes
  - año

En lenguaje C las cadenas de caracteres tendrán un máximo de 100 caracteres.

Los posibles valores de la variable que guarda el sexo son *boy* o *girl*.

El código del centro es un valor numérico que está dentro del rango de 1 a 99999.

Para la fecha se necesita tener por separado el día, mes y año.

En lenguaje algorítmico:

```
type
  tSex = { Boy, Girl }
end type

type
 tDate = record
    day: integer;
    month: integer;
    year: integer;
 end record
end type

type
  tStudent = record
    name: string;
    lastnames:  string;
    sex: tSex;
    isCeliac: boolean;
    haveSiblingsInThisSchool: boolean;
    schoolCode: integer;
    inscriptionDate: tDate;
  end record
end type

var
  student: tStudent;
end var
```

En lenguaje C:

```
/* Constants definition */
#define MAX_LENGTH 100

/* Type definition */
typedef enum { false, true } bool;
typedef enum { Boy, Girl } tSex;

typedef struct
{
    unsigned int day;
    unsigned int month;
    unsigned int year;
} tDate;

typedef struct
{
    char name[MAX_LENGTH];
    char lastnames[MAX_LENGTH];
    tSex sex;
    bool isCeliac;
    bool haveSiblingsInThisSchool;
    unsigned long int schoolCode;
    tDate inscriptionDate;
} tStudent;

/* Variable definition */
tStudent student;
```

b) Un tipo **tGame** para clasificar cada uno de los juegos de consola de un centro comercial.

Se necesita almacenar:

- nombre del juego
- tipo de juego
- edad mínima recomendada para utilizarlo
- un vector donde guardar todos los tipos de consolas para los que existe el juego
- precio del juego

En lenguaje C las cadenas de caracteres tendrán un máximo de 100 caracteres.

Los posibles tipos de juego son *aventura, carreras, deporte, estrategia, simulación* o *musical.*

Los posibles tipos de consola son: *ps, nin, cub, xb*.

En lenguaje algorítmico:

```
type
  tGenre = { Adventure, Racing, Sport, Strategy, Simulation, Musical }
end type

type
  tConsole = { PS, NIN, CUB, XB }
end type

type
  tGame = record
    name: string;
    genre: tGenre;
    recommendedMinimumAge: integer;
    consoles: vector of tConsole;
    price: integer;
  end record
end type

var
  game: tGame;
end var
```

En lenguaje C:

```
/* Constants definition */
#define MAX_LENGTH_NAME 100
#define MAX_LENGTH_CONSOLES 4

/* Type definition */
typedef enum { Adventure, Racing, Sport, Strategy, Simulation, Musical } tGenre;
typedef enum { PS, NIN, CUB, XB } tConsole;

typedef struct
{
    char name[MAX_LENGTH_NAME];
    tGenre genre;
    unsigned char recommendeMinimumAge;
    tConsole consoles[MAX_LENGTH_CONSOLES];
    unsigned int price;
} tGame;

/* Variable definition */
tGame game;
```

[15%] Ejercicio 2: Acciones y funciones

Resuelve tanto en **lenguaje algorítmico** como en **lenguaje C.**

Escribe la cabecera correspondiente a la acción o función que realiza cada una de las siguientes tareas.

Recuerda que únicamente se pide la cabecera, **no** hay que implementar ninguna línea de código.

En lenguaje C podéis suponer que tenéis definido el tipo *tBool* para almacenar booleanos.

a) translateText

Acción que lea un texto escrito en inglés de la entrada de datos estándar, y escriba el mismo texto traducido al francés por la salida estándar de datos.

```
action translateText(textToTranslate: string);
```

```
void translateText(char textToTranslate[]);
```

b) isLeapYear

Función que dado un año devuelva si se trata o no de un año bisiesto.

```
function isLeapYear(year: integer) : boolean;
```

```
tBool isLeapYear(unsigned int year);
```

c) updateAccount

Acción que dado el:

- **total del extracto** de una cuenta bancaria a fecha **30/01/1999**
- y el **gasto realizado** durante el día **31/01/1999**
- **retorna** el **extracto total** actualizado a fin de mes

```
action updateAccount(currentBalance: real, payout: real, inout balanceUpdated: real);
```

```
void updateAccount(float currentBalance, float payout, float *balanceUpdated);
```

[20%] Ejercicio 3: Modularidad

Resuelve únicamente en **lenguaje algorítmico.**

Dado el siguiente tipo de datos estructurado correspondiente a la definición de las coordenados 2D de un punto:

```
type
    tPoint : record
        x: real;
        y: real;
    end record
end type
```

- Define la **cabecera** e **implementa** el código de una **acción** (***readPoint***) que:
  - lea de la entrada estándar las **coordenadas** de un punto
  - y **retorne** la información leída en una variable de tipo ***tPoint***.

```
action readPoint(inout point: tPoint)
      point.x := readReal();
      point.y := readReal();
end action
```

- Define la **cabecera** e **implementa** el código de una **acción** (***writePoint***) que
  - **escriba** por la salida estándar las coordenadas de un punto guardado en una variable de tipo *tPoint.*

```
action writePoint(point: tPoint)
      writeReal(point.x);
       writeReal(point.y);
end action
```

- Define la **cabecera** e **implementa** el código de una **función** (***equalCoord***) que
  - dada una variable de tipo *tPoint*
  - **devuelva** *true* si la coordenada *x* es igual a la coordenada *y*
  - ***devuelva*** *false* en caso contrario.

```
function equalCoord(point: tPoint) : boolean
      if point.x = point.y then
            return TRUE;
      else
            return FALSE;
end function
```

- Implementa un **algoritmo** que:
    1. Lea por el canal de entrada estándar las coordenadas de un punto y las guarde en una variable de tipo *tPoint*.
    2. Escriba por el canal de salida estándar las coordenadas del punto leído.
    3. Escriba por el canal de salida estándar el texto *"Equals"* si la coordenada x es igual a la coordenada y, y el texto *"Different"* en caso contrario.

```
algorithm workingWithPoints
        { variables declaration }
        var
                point: tPoint;
                areEquals: boolean;
        end var

        readPoint(point);

        { write point }
        writePoint(point);

        { check coordinates }
        areEquals := equalCoord(point);

        if areEquals then
                writeString("Equals");
        else
                writeString("Different");

end algorithm
```

[40%] Ejercicio 4: Modularidad

Resuelve únicamente en **lenguaje C.**

En este ejercicio trabajaremos la modularidad en dos sentidos. Primeramente, dividiendo el código en acciones y funciones que nos permitan poder reutilizarlo, y por otro, trabajar con diferentes archivos.

Nos planteamos una petición para organizar los estudiantes de un curso con los diferentes perfiles de profesores del centro. Para resolver este ejercicio, planteamos los siguientes apartados:

Puedes crear las constantes que creas necesarias.

Puedes definir e implementar las acciones y/o funciones adicionales que creas necesarias.

a) Crea en *Codelite* un nuevo espacio de trabajo y dentro un nuevo proyecto denominado **StudentsTeachers**.

Para organizar el proyecto crearemos (tanto físicamente como en el proyecto) una carpeta **include** (donde guardar los archivos de cabecera) y una carpeta **src** (donde guardar la implementación).

Inicialmente reubica físicamente el archivo **main.c** en el directorio **src** y verifica que el proyecto compila correctamente.

Nota: Recuerda que en la Wiki de la asignatura dispones de un video explicativo de cómo crear y organizar los proyectos de Codelite.

Es **obligatorio** que la solución propuesta siga la estructura indicada.

b) Crea un nuevo archivo **StudentsTeachers.h** en el directorio **include**, y un nuevo archivo **StudentsTeachers.c** en el directorio **src**.

c) Define en el archivo **StudentsTeachers.h** cuatro tipos de datos estructurados:

1. Tipo de datos **tTeacher** que represente la información de un profesor:
   a. Entero positivo **idTeacher** (el rango de valores irá de 1 a 10).
   b. Booleano **bEnglish**.
   c. Carácter **cLevel**.
2. Tipo de datos **tTeacherVector** que guarde un vector de 4 elementos de tipo **tTeacher**.

3. Tipo de datos **tStudent** que represente la información de un estudiante:
    a. Entero positivo **dni** (el rango de valores irá de 1 a 999).
    b. Booleano **bEnglish**.
    c. Carácter **cLevel**.
    d. Entero positivo **idTeacher**.
4. Tipo de datos **tStudentVector** que guarde un vector de 6 elementos de tipo **tStudent**.

d) Implementa la acción **fillTeachersVector** que tiene que devolver una variable de tipo **tTeacherVector** inicializada con los valores que se indican en la siguiente tabla:

Posición vector	IdTeacher	bEnglish	cLevel
0	1	true	P
1	2	false	P
2	3	true	S
3	4	false	S

Recuerda que tienes que definir la cabecera en el fichero **StudentsTeachers.h** y la implementación en el fichero **StudentsTeachers.c**.

Puedes crear e implementar las acciones adicionales que creas necesarias.

e) Implementa la acción **readStudent** que lea del canal de entrada estándar parte de la información de un estudiante y la guarde en una variable de tipo **tStudent**. Tiene que leer:

1. Identificador numérico que guardaremos en la variable **dni**.
2. Valor numérico 0 o 1 indicando si las clases las hará o no en inglés (valor 0 indica que no es inglés y valor 1 indica que es inglés). Valor a guardar en la variable **bEnglish.**
3. Carácter indicando el nivel. Los valores podrán ser 'P' o 'S'. Valor a guardar en la variable **cLevel**.

Recuerda que tienes que definir la cabecera en el fichero **StudentsTeachers.h** y la implementación en el fichero **StudentsTeachers.c**.

Puedes presuponer que los valores introducidos por el usuario serán correctos.

Recuerda que antes de leer un carácter es necesario llamar a la función **getchar();**

f) Implementa la acción **writeStudent** que dada una variable de tipo **tStudent** escriba por el canal de salida estándar el valor de las variables **dni** y **idTeacher**.

Recuerda que tienes que definir la cabecera en el fichero **StudentsTeachers.h** y la implementación en el fichero **StudentsTeachers.c**.

g) Implementa una acción **updateTeacherStudent** que dada una variable de tipo **tTeacherVector** y una variable de tipo **tStudent** devuelva esta última variable con el valor de **idTeacher** actualizado.

La acción tiene que buscar en el vector **tTeacherVector** el identificador de profesor que le corresponde al estudiante según el nivel y si realiza o no las clases en inglés. Si no encontramos ningún profesor adecuado al estudiante, guardaremos el valor -1 en la variable **idTeacher**.

h) Modifica la función **main** tal y como se indica a continuación.

Puedes crear e implementar las acciones adicionales que creas necesarias.

1. Crea una variable de tipo **tTeacherVector** y llama a la acción **fillTeachersVector** para inicializar la variable.
2. Crea una variable de tipo **tStudentVector** y realiza de forma iterativa los siguientes pasos para rellenar y actualizar todas las posiciones del vector de estudiantes:
    i. Utiliza la acción **readStudent** para pedir la información de un estudiante y guardarla en el vector de estudiantes.
    ii. Utiliza la acción **updateTeacherStudent** para actualizar el identificador de profesor para el estudiante leído anteriormente.
3. Recorre todas las posiciones del vector de la variable **tStudentVector** y utiliza la acción **writeStudent** para escribir la información necesaria de cada uno de los estudiantes.