ANSI C Like OOP

Programación I – Laboratorio I.
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Esp. Lic. Mauricio Dávila*

Revisores: *Esp. Ing. Ernesto Gigliotti*

Versión : 1.2

Índice de contenido

1 Funciones asociadas a una entidad

Nos referimos al conjunto de funciones que forman parte de la base de cada entidad y se deben desarrollar para cada una independientemente de cuál sea el propósito específico de la entidad, por ejemplo funciones del estilo:

entidad_new()
entidad_newParametros()
entidad_delete()
entidad_setPropiedad()
entidad_getPropiedad()
isValidPropiedad()

1.1 Entidad

Una entidad es la representación de un objeto o concepto del mundo real, en ANSI C las entidades se modelizan mediante estructuras donde los miembros o campos de la misma normalmente representan los atributos o propiedades de dicha entidad. En el siguiente ejemplo se declara la entidad Alumno la cual tiene como propiedades: nombre, altura e id.

```
typedef struct
{
	char nombre[50];
	float altura;
	int id;
}Alumno;
```

El campo **id** debe permitir identificar a cada elemento de la entidad de manera unívoca, en el caso del ejemplo, esto implica que no podrán existir dos Alumnos con el mismo **id**. La manera de asegurar que este campo es unívoco será no permitiendo que éste sea manipulado directamente por el usuario como se verá más adelante.

1.2 Constructor

Los constructores son funciones que sirven para reservar memoria dinámica para un elemento de la entidad que construyen.

Tienen por nombre **entidad_new**
Tienen como tipo de retorno un puntero de la entidad que construyen
No recibe parámetros
Deben ser públicos, no tendría ningún sentido declarar un constructor como privado.

```
Alumno* alumno_new(void)
{
	return (Alumno*) malloc(sizeof(Alumno));
}
```

1.3 Constructor con parámetros

Los constructores con parámetros son funciones que sirven tanto para reservar memoria dinámica para un elemento de la entidad que construyen, como para darle valor inicial a uno o varios campos de dicha entidad.

Tienen por nombre **entidad_newParametros**

Tienen como tipo de retorno un puntero de la entidad que construyen

Reciben como parámetro todos los campos de la entidad que construyen

Deben ser públicos, no tendría ningún sentido declarar un constructor como privado.

```
Alumno* alumno_newParametros(char* nombre,float altura,int id)

{
	Alumno* this = alumno_new();

	if(	!alumno_setNombre(this,nombre) &&
		!alumno_setAltura(this, altura) &&
		!alumno_setId(this, id))
	{
		return this;
	}
	alumno_delete(this);
	return NULL;
}
```

1.4 Destructor

Los destructores a diferencia de los constructores, que se encargan de reservar memoria dinámica para un elemento de la entidad, se dedican a liberar la memoria previamente reservada.

Tienen por nombre **entidad_delete**

No retornan valor

Recibe como parámetro el puntero al elemento de la entidad que se desea eliminar

Deben ser públicos, no tendría ningún sentido declarar un destructor como privado.

```
void alumno_delete(Alumno* this)
{
	free(this);
}
```

1.5 Setters

Del Inglés *set*, que significa establecer, permite asignar un valor validado a un campo de la entidad.

Tienen por nombre **entidad**_*set***Atributo**

Retorna como valor un entero que indica el código de error

Recibe como parámetro el puntero al elemento de la entidad al que se le desea setear un atributo y el valor del atributo que se desea setear

Deben ser públicos, no tendría ningún sentido declarar un *setter* como privado

```
int alumno_setNombre(Alumno* this, char* nombre)
{
	int retorno = -1;
	if(this != NULL && nombre != NULL && isValidNombre(nombre))
	{
		retorno = 0;
		strcpy(this->nombre,nombre);
	}
	return retorno;
}
```

1.6 Getters

Del Inglés *get*, que significa obtener, permite acceder al valor de un atributo de la entidad.

Tienen por nombre **entidad_getAtributo**

Retorna como valor un entero que indica el código de error

Recibe como parámetro el puntero al elemento de la entidad al que se le desea leer un atributo y el puntero a ser utilizado para retornar valor del atributo

Deben ser públicos, no tendría ningún sentido declarar un *getter* como privado

```
int alumno_getNombre(Alumno* this, char* nombre)
{
	int retorno = -1;
	if(this != NULL && nombre != NULL)
	{
		retorno = 0;
		strcpy(nombre,this->nombre);
	}
	return retorno;
}
```

1.7 setId

Para lo cual se emplea una variable del tipo estática (**static**), este tipo de variables persiste cuando la función termina (es decir cuando es eliminada de la pila de llamadas) y al ser ejecutada nuevamente sigue manteniendo su valor.

Tienen por nombre **entidad**_*set***Id**

Retorna como valor un entero que indica el código de error

Recibe como parámetro el puntero al elemento de la entidad al que se le desea setear el id y id que se desea setear. Al recibir un id mayor o igual a cero verifica si se trata de un nuevo máximo, de ser así lo conserva y en el caso de recibir un id negativo, calcula el id a asignar automáticamente.

Deben ser públicos, no tendría ningún sentido declarar un *setter* como privado

```
int alumno_setId(Alumno* this, int id)
{
	int retorno = -1;
	static int maximoId = -1;
	if(this != NULL)
	{
		retorno = 0;
		if(id < 0)
		{
			maximoId++;
			this->id = maximoId;
		}
		else
		{
			if(id > maximoId)
				maximoId = id;
			this->id = id;
		}
	}
	return retorno;
}
```

1.8 isValidAtributo

Este tipo de funciones serán las encargadas de validar cada uno de los atributos de la entidad, deberá de existir una para cada atributo. Si una función es declarada como estática (**static**), solamente podrá accederse a ella desde el archivo en que fue declarada. Esto es lo que haremos con este tipo de funciones, ya que solo necesitan ser usadas internamente.

Tienen por nombre isValid**Atributo**

Retorna como valor un entero que indica 1 si el atributo es válido y 0 si no lo es,

Recibe como parámetro el atributo

Deben ser privadas ya que solo se utilizan desde el archivo de la propia entidad

Los prototipo de estas funciones no forman parte del archivo de header (*.h) se deben colocar en el archivo de código (*.c)

```
static int isValidNombre(char* nombre)
{
	if(strlen(nombre) < 50 && validation_isValidName(nombre))
		return 1;
	return 0;
}
```