Guía de estilo

A modo de guía general a satisfacer en todas las entregas, se muestra lista de criterios a aplicar a la hora de codificar código en las prácticas de la asignatura “Sistemas operativos”:

1. Estructura del programa

La estructura básica de un programa de un solo módulo ha de constar de los siguientes elementos y en el siguiente orden:

- Una cabecera
- Instrucciones #include: primero las inclusiones de bibliotecas del sistema, tras ello se incorporan las bibliotecas definidas localmente
- Instrucciones #define
- Instrucciones typedef
- Prototipos
- Variables globales (se debe evitar su uso)
- Función main
- Resto de funciones

Además, se debe crear una cabecera de comentarios (siguiendo la sintaxis de Doxygen) para documentar cada módulo y cada función

2. Convención sobre nombres

	Convención	**Ejemplo**
funciones	En minúsculas (lowercase) Si el nombre consta de más de una palabra, separarlas mediante “_”	seleccionar_salida
variables	En minúsculas (lowercase) Si el nombre consta de más de una palabra, separarlas mediante “_”	n_elementos
variables *anónimas*	Una sola letra en minúscula	i++;
nueva declaración de tipos mediante typedef	Criterio CamelCase	typedef struct _tipoLista TipoLista; typedef struct _position { int x; int y; } position;
constantes	MAYÚSCULAS (UPPER_CASE), palabras separadas mediante “_”	#define MAX_LENGTH 5

3. Plantilla

- Indentar mediante espacios en lugar de usar el tabulador

- Utilizar 3 ó 4 espacios de indentación por nivel, pero no mezclar el criterio en un mismo fichero
- Delimitar bloques de código mediante indentación y no sólo en base a {}
- Se puede incluir un espacio antes y/o después de los paréntesis que delimitan la lista de argumentos de una función
- Escribir los corchetes que delimitan el cuerpo de una función de modo compacto, según figura en el siguiente ejemplo

```
void process_input ( int age ){
        printf("Process!!!!");
}
```

- Usar siempre {} incluso para bloques de una línea de código
- Situar el corchete de apertura de una condición de acuerdo con el criterio de compactación, es decir, justo después de la condición en lugar de ubicarlo al principio de la línea siguiente. Esto es, seguir el criterio ilustrado mediante el siguiente ejemplo:

```
while ( cond == 0 ) {
        …;
}
```

- Aplicar el criterio de compactación de código a los corchetes empleados en las cláusulas else, de acuerdo con el siguiente ejemplo:

```
if ( cond == 0 ) {
        …;
        …;
}   else {
        …;
}
```

- No indentar else if and else cuando aparecen encadenadas en una estructura como la que sigue

```
if ( cookies ==0 ) {

        …;

} else if ( cookies  <  10 ) {

        …;

} else if (cookies  <  50 ) {

        …;

} else  {

        …;

}
```

4. Declaraciones

- Las variables deben ser declaradas al comienzo de la función
- En el caso de que se inicialicen variables, se declararán una por una en una línea por separado. Sólo se emplearán lista de variables separadas por coma cuando se efectúe una declaración de un conjunto de variables relativas a una misma categoría o criterio (e.g., int row, col;)
- En un bloque de definiciones relacionadas, alinear verticalmente los nombres, los tipos y las inicializaciones
- Inicializar las variables cerca de donde son usadas por primera vez
- Todas las estructuras (struct) y enumerados deben ser definidos mediante typedefs
- Situar al comienzo del fichero los prototipos de las funciones que van usadas sólo localmente
- Situar al comienzo del fichero los typedefs para tipos que sólo van a ser usados localmente
- Los prototipos y las definiciones de tipos que van a ser usados en más de un fichero debe ser emplazados en un fichero .h que es nombrado con un nombre adecuado. Este fichero será incluido (mediante #include) en el fichero en el que la función relativa es definida, y además en cada uno de los ficheros que usan la función o tipo
- No incluir mediante un fichero de cabecera .h en otro fichero .h

5. Funciones

- Escribir el tipo devuelto por una función en la misma línea que el nombre de la función
- Los cuerpos de las funciones deben ser compactos, de forma que se favorecerá la división de funciones en sub-funciones si el cuerpo es excesivamente largo
- Cada función debe tener un cometido específico y bien definido, el cual debe estar bien referido mediante el nombre que se le asigna a la función
- La primera operación que ser realiza en una función es la verificación de los argumentos de entrada
- Evitar el uso de macros
- Evitar el uso de variables globales: son fuente de efectos laterales (side effects)
- Asignar nombres a las variables/constantes/funciones de modo consistente,
- La inclusión de comentarios en el código debe seguir la premisa de brevedad, y sólo serán incluidos para explicar elementos o casuísticas de difícil comprensión