Estos apuntes están en continua construcción.

Programming is best regarded as the process of creating works of literature, which are meant to be read.
-- Donald Knuth, Literate Programming

Introducción

El código fuente se tiene que escribir para que sea fácil de leer por otros desarrolladores. Recordar que éste se inventó por nuestra incapacidad para comunicar instrucciones en binario a la máquina. Dos programas pueden ser funcionalmente equivalentes para un mismo conjunto de entradas pero diferenciar sensiblemente en el estilo de programación en que han sido descritos. Así, un buen estilo representa un indicador de tu profesionalidad.

Entregables

- Incluir **Makefile** (véase sección Makefile).

- Respecta la nomenglatura definida para el nombre del fichero que se entrega. En particular, proveer un fichero con la compresión que se indica. No se proveerán ficheros .rar.

- La documentación se proveerá en alguno de los siguientes formatos: **texto plano**, **html** o **pdf**.

- No se incluirá archivos o directorios propietarios de un entorno de desarrollo en particular.

Estilo de programación

- Código **correctamente indentado (**recomendación: **4 espacios)**. Cualquier editor decente tiene la posibilidad configurar para que se inserten espacios en vez de tabuladores, y configurar el número de espacios que se incluyen cada vez que se tabula. Las líneas, tanto de código como de comentario, no deberían superar los **80 caracteres**.

- La codificación de caractares en **UTF-8**.

- **Realizar un sangrado correcto del programa.** Hay que asegurarse que la indentación de las sentencias refleje fielmente la estructura del programa, de manera que sea fácil de identificar si un bloque se encuentra dentro de otro.

- Nombres descriptivos para variables globales, nombres cortos para locales.

- Nombres de variable y funciones en minúsculas.

- Constantes en mayúsculas.

- **Respetar los giros idiomáticos propios del lenguaje** que han sido establecidos por el uso que hacen del mismos la comunidad de desarrolladores expertos. Existen una serie de convenios a la hora de nombrar variables que están bastante extendidos. Estos son útiles cuando no se tiene un nombre específico para una variable
    - Variables temporales: tmp o aux.
    - Contadores: i, j, k, l …
    - Número enteros: a,b,c …
    - Números reales: x, y ,z
    - Carácter: c
    - Cadena de caracters: s, str

Ejemplo incorrecto:
int cont1, cont2

- Ser consistente (véase TPoP). Es importante mantener una cierta coherencia en la elección de los nombres de variable. De esta manera, será más fácil entender para qué sirve una variable y su relación con el resto.

- Usar verbos para las funciones: verbo + sustantivo (véase TPoP). Un criterio que nos puede ayudar a nombrar un función es emplear un verbo en activo + un sustantivo que indique la acción que realiza la función. Ej, esPrimo(n), leeAlumnos(tabla), calculaDistancia(p1, p2) ó calDistancia(p1,p2)

- Economía de variables. Intentar ajustar el número de variables que se emplean. **Nunca dejes declaradas variables en el código que no se empleen**. Un par de criterios que nos pueden ayudan a decidir si necesitamos una variable o no:
    - Si una misma expresión se emplea para calcular un valor en determinados puntos del código, mejor emplear una variable para realizar el cálculo una sola vez.
    - Si mejora la legibilidad del código el realizar cálculos intermedios empleando más variables.

- Asegurarse que todas las variables tienen un valor inicial antes de utilizarlas. Hay que revisar el código para comprobar que todas las variables tienen asignado un valor (ya sea porque se lee, o porque se calcula) antes de emplearlas. **No asumas que las variables se inicializan a un valor por defecto** (ej, cero).

- Declaración de constantes. En general, las constantes se nombran siguiendo los mismo criterios que una variable global, pero con el nombre en mayúsculas. De esta manera será sencillo distinguirlas en el código. **Todos los números mágicos del código tendrán que sustituirse por constantes**.

Ejemplo incorrecto	Ejemplo correcto
dst[i] = 32;	dst[i] = ‘ ‘;

Tampoco se renombran constante que ya existen:

Ejemplo incorrecto	Ejemplo correcto
#define ASCII_ESPACIO 32	Se emplea el caracter constante ‘ ‘

- Los *imports* se añadirán al principio del programa en orden alfabético. Primero aquellos que hacen referencia a bibliotecas estandar, y luego las bibliotecas propias.

- Código inconsistente: Ej. se comprueba algunas veces el retorno de una función y otras veces no.

- **Limitar el uso de macros**.

- **Duplicidad del código (bucles y funciones).** Evitar la duplicidad de código. Siempre que en un programa exista un fragmento de código que se repita un número determinado de veces, dependiendo del caso concreto:
    - Tratar de convertir ese código en una función que se invoque siempre que sea necesario.
    - Convertir el código en el cuerpo de un bucle que se repite tantas veces como sea necesario.

Compilación y construcción

- El código tiene que compilar **sin** producir advertencias (warnings).

- Opciones de compilación básicas: -Wall -ansi -pedantic
Es obligatorio incluir la opción -Wall
- Opciones de compilación de depuración: -g
- Opciones de compilación en producción: -O3

Ejecución

- Es necesario comprobar que el número de argumentos del programa es correcto.

- La ayuda de los programas tiene que ser lo más explicativa posible: **causa del error** y **formato correcto**. Emplear `perror` para errores de llamadas al sistema.

- Se tiene que respetar **escrupulosamente** el formato de entrada y salida especificado.

Makefile

- Tiene que incluir un objetivo `all` (que construya todos los ejecutables) y un objetivo `clean` que borre los archivos temporales.

- Tiene que incluir la construcción de todos los ejecutables y clientes de prueba.

- Tiene que reconstruir y recompilar aquellos archivos fuente que hayan cambiado, y sólamente estos.

- Tiene que incluir las opciones de compilación (CFLAGS) y enlazado (LDFLAGS), si las hubiera, en variables (véase compilación y construcción)

Comentarios

- **¡No comentar lo obvio**! (véase TPoP). El propio código que escribimos debería ser en gran medida autoexplicativo. Si una parte del código requiere un comentario, intentar comentar bloques, no sentencias aisladas. Tan peligroso es no comentar nada, como comentar en exceso. Pensar siempre si es necesario reescribir el código, antes de comentar un código mal escrito.

Incorrecto:
longitud = strlen(ori); /*Calcular el tamaño de la cadena*/

- Comentar funciones y variables globales
Cada función o procedimiento debe tener un comentario de cabecera con información de:

- Qué acción realiza.
- Sus parámetros de entrada y de salida, así como el significado de cada uno de ellos.
- El valor devuelto.
- Los requisitos o precondiciones en los valores de los parámetros.
- Las condiciones de error tratadas dentro de cada acción.

- **No contradecir el código** (véase TPoP). Hay que tener cuidado de mantener coherentes los comentarios con el código. Es posible que se cambie el código y no se actualice el comentario, lo que da lugar a confusión cuando se vuelve a revisar el código pasado un tiempo. Un ejemplo típico de este tipo de error son las cabeceras de las funciones y procedimientos.

- Las funciones de la biblioteca estándar se presumen su funcionamiento es conocido

Incorrecto:
toupper(ori[i]); /* la funcion toupper convierte a mayusculas la cadena */

- Emplear /* */ en vez de //

Uso del if

- **Emplear if-else en vez de if-if**. Si tenemos dos condiciones de manera que una es la contraria de la otra, se empleará if-else, en vez de realizar dos if.

Ejemplo incorrecto	Ejemplo correcto

`if (distancia <= radio) {`
`  puts("PUNTO INTERIOR");`
`}`
`if (distancia > radio) {`
`  puts("PUNTO EXTERIOR");`
`}`
	`if (distancia <= radio) {`
	`  puts("PUNTO INTERIOR");`
	`} else {`
	`  puts("PUNTO EXTERIOR");`
	`}`

- **Anidar if:** Esta regla es una generalización de la anterior. Si tenemos tres o más condiciones autoexcluyentes se empleará if-else if, en vez de realizar varios if.

- **Condición en positivo en vez de en negativo**: En general, queda más claro el código si se comprueba que se cumpla que la condición sea verdadera en vez de comprobar que sea falsa.

Ejemplo incorrecto	Ejemplo correcto

`if (! distancia <= radio) {`
`  puts("PUNTO EXTERIOR");`
`} else {`
`  puts("PUNTO EXTERIOR");`
`}`
	`if (distancia <= radio) {`
	`  puts("PUNTO INTERIOR");`
	`} else {`
	`  puts("PUNTO EXTERIOR");`
	`}`

- **Extraer partes comunes**. Aquellas partes que sean comunes a las dos ramas que define la condición deberían sacarse fuera de la sentencia condicional en vez de repetirse en cada rama.

Ejemplo incorrecto	Ejemplo correcto

`if (x > y) {`
`  max = x;`
`  printf("max: %d\n", x);`
`} else {`
`  max = y;`
	`if (x > y) {`
	`  max = x;`
	`} else {`
	`  max = y;`
	`}`

`    printf("max: %d\n", y);` `}`	`printf("max: %d\n", max);`

Ejemplo incorrecto	Ejemplo correcto

`if (x > y) {`
`    max = x;`
`    printf("max: %d\n", max);`
`}`
`printf("max: %d\n", max);`
	`if (x > y) {`
	`    max = x;`
	`}`
	`printf("max: %d\n", max);`

Uso de sentencias de repetición

- En un bucle `for` **no se modifica la variable de iteración dentro del bucle**.

Ejemplo incorrecto	Ejemplo correcto

`for (i = 1; i <= MAX; i++) {`
`    printf("i: %d\n", i);`
`    i++;`
`}`
	`for (i = 1; i <= MAX/2; i++) {`
	`    printf("i: %d\n", 2*i-1);`
	`}`

- En general, los bucles definidos se implementan utilizando `for`, y los bucles indefinido empleando `while` o `do..while`.

Ejemplo incorrecto	Ejemplo correcto

`for (i = 0; i < N; i++) {`
`    /* Más codigo */`
 	
`    if (res > UMBRAL) {`
`        break;`
`    }`
`}`
	`while (i < N && res > UMBRAL) {`
	`    /* Más codigo */`
	`}`

`while (i < N) {`
`    /* Más codigo */`
`    i++;`
`}`
	`for (i = 0; i < N; i++) {`
	`    /* Más codigo */`
	`}`

- Indicar todas las condiciones de mantenimiento en las condiciones del bucle. Si es necesario incluir banderas para identificar el momento en el que debe finalizar el bucle.

Ejemplo incorrecto	Ejemplo correcto

`while (i < N) {`
`    /* Más codigo */`
`    if (ganado == TRUE) {`
`        break;`
`    }`
`}`
	`while (i < N && ganado != TRUE) {`
	`    /* Más codigo */`
	`}`

Funciones

- **Sentencias de escritura dentro de funciones**. Si una función/procedimiento no se emplea específicamente para imprimir datos, no incluyáis sentencias de escritura de datos (write, writeln). Si no, cuando se quiera emplear la función en un programa distinto al que se había concebido, se incluirá en la salida del programa texto indeseado. EXCEPCIÓN: por motivos de depuración, pero siempre deshabilitándolas en la versión definitiva.

- Funciones privadas se declaran como `static`.

- Aquellos argumentos cuyo valor no se modifica se declaran con `const`.

Gráficas

- Las etiquetas tienen que ser legibles.

- Escalar los ejes de manera que se ocupe el máximo espacio de la gráfica.

- Se tiene que incluir los títulos de los ejes, y las unidades en qué están medidos.

- La leyenda de los ejes tiene que incluir una descripción significativa de los mismos.

- La comparación de varios algoritmos se debe visualizar sobre la misma gráfica, no en gráficas individuales.

Bibliografía

TPoP: The Practice of Programming by Brian W. Kernighan and Rob Pike. Addison-Wesley, Inc., 1999. ISBN 0-201-61586-X.