Guía de estilo y buenas prácticas de programación en C/C++

Introducción

A la hora de escribir código fuente en un determinado lenguaje de programación es aconsejable seguir unas guías de estilo. Esto te permitirá escribir el código de una manera homogénea, lo cual facilitará su modificación y revisión posterior tanto por el programador que lo ha escrito como por otros programadores distintos.

La regla más importante es la siguiente: sé coherente con tu propio estilo, escribe lo mismo siempre de la misma manera. Ten tu propia guía de estilo, ahorrarás mucho tiempo al escribir código y sobre todo al revisarlo. Esta guía pretende orientarte sobre la manera de escribir código, dándote a conocer determinadas convenciones y buenas prácticas recomendables.

Estructura del fichero fuente

En las prácticas de programación debes respetar la siguiente estructura de código fuente en todos los ficheros que contengan código (`.cc`):

- Todos los ficheros (incluídos los de cabecera, pero sin incluir el `makefile`, utilizado en Programación 2) comenzarán con un comentario en la primera línea con el DNI y el nombre del alumno a quien pertenece la práctica, de la siguiente manera:

  ```
  // DNI 123456789 LOPEZ LOPEZ, JUAN
  ```

- A continuación se incluirán los ficheros de cabecera con la directiva `#include`, primero los ficheros del sistema, que van con `<>`, a continuación la orden:

  ```
  using namespace std;
  ```

  y luego los ficheros de cabecera propios del programa, que van con `""`. Tanto en los ficheros de cabecera (`.h`) como en los ficheros de código (`.cc`), *solo se debe incluir con* `#include` *aquellos ficheros de cabecera que sea necesario para compilar el fichero.*

- Si es necesario, se declararán constantes y tipos que se utilicen únicamente en el fichero.

- Por último, el fichero incluirá el código de las funciones correspondientes.

Tamaño de las funciones

Por muy bien escrita que esté una función, si ocupa más de 20-30 líneas obliga al lector a hacer scroll para verla completa, y eso puede dificultar su comprensión. En general, salvo casos excepcionales (por ejemplo, cuando se utilizan sentencias switch con muchos casos), es recomendable que las funciones sean pequeñas y se puedan ver de un sólo vistazo, sin tener que hacer scroll para verlas completas. Además, cuando una función *crece* demasiado quizá sea el momento de plantearse hacer otra u otras funciones que realicen alguna subtarea de las que tiene que hacer la función.

La función `main` es una función como las demás, y por tanto debe tener un tamaño razonable, pero sin *pasarse*:

```
int main(){
  principal();    // MAL

  return 0;
}
```

Nombres de variables, declaraciones e inicializaciones

Los nombres de variables deben ser representativos de su contenido. Irán en minúscula.

Realiza una única declaración de variable por linea, de esta manera facilitas comentar para qué vas a usar la variable, y si es posible inicializa la variable cuando la declares. Los nombres de tipo, los identificadores y los comentarios estarán alineados en la manera de lo posible.

```
char tablero[SIZE][SIZE];  // declaro el tablero de juego
int  i = 1; // índice para recorrer las filas
```

Para los identificadores de función utiliza *wordmixing* (mezcla de palabras) con la primera letra de cada palabra en mayúsculas. La primera palabra deberá ser un verbo que refleje lo que hace la función y la segunda palabra el objeto sobre el que se aplica la acción. Por ejemplo:

```
void mostrarTablero (char tablero[][SIZE]);
```

Constantes

Como en el caso de variables y funciones, el nombre de las constantes debe ser significativo y debe indicar para qué se usa la constante, no su valor. Es conveniente usar letras mayúsculas para el nombre. Otro estilo que se suele emplear es utilizar una palabra que empiece por la letra K :

```
const int CERO=0;           // NOMBRE MAL ELEGIDO
const int MAXCLIENTES=1000;
const int KMAXVIDEOS=400;
```

Por otro lado, no todas las constantes que aparezcan en un programa deben ser declaradas explícitamente como constantes por el programador, por ejemplo, el 0 se utiliza muy a menudo para recorrer vectores y matrices, y no se debe declarar como constante por ello. Sin embargo, si nos dicen que un valor vacío en un conjunto de datos numéricos se va a representar con un 0, en ese caso debemos declarar una constante cuyo valor sea 0 (por si en un futuro el valor vacío se representa con un -1, por ejemplo):

```
const int VACIO=0;
```

En general, solamente aquellos datos que puedan ser modificados en futuras versiones del programa (debido a cambios en la especificación), deben ser declarados como constantes. Por ejemplo, si debemos hacer un programa para mantener una lista de alumnos de una asignatura y nos dicen que como máximo va a haber 100 alumnos, el valor 100 debe ser declarado como constante puesto que a pesar de ser un valor constante durante la ejecución del programa, podría cambiar en futuras versiones.

Tipos

El nombre de los tipos debe ser significativo y conviene que comience por la letra T:

```
typedef int TVector[10];

TVector notas;  //notas es un vector de 10 elementos
```

Sangrado de código

No existe un único tipo de sangrado de código (o indentación) en el lenguaje C/C++. Este lenguaje es de sangrado libre, y el sangrado no denota sintaxis como en otros lenguajes, sino que se usa únicamente por legibilidad.

Existen básicamente tres tipos de sangrado en C, y son los siguientes:

- Sangrado de Kernighan & Ritchie: la llave de apertura de bloque va en la misma linea que la condición de la sentencia, y la de cierre va al nivel de indentación de la sentencia. Ejemplo:

```
if (a > 5) {
  // estilo K&R
}
```

- Estilo ANSI C++: la llave de apertura de bloque va en la siguiente linea, y tanto esta llave como la de cierre van al nivel de indentación de la sentencia. Ejemplo:

```
if (a > 5)
{
  // Estilo ANSI C++
}
```

- Estilo GNU: la llave de apertura de bloque va en la siguiente linea, y tanto esta llave como la de cierre van indentadas respecto a la sentencia. Ejemplo:

```
if (a > 5)
  {
    // Estilo GNU
  }
```

No hay un estilo de sangrado mejor que otro, pero debes intentar usar siempre el mismo por claridad. En las asignaturas de programación recomendamos el uso de los estilos K&R o ANSI C++.

No es recomendable que uses espacios para realizar el sangrado sino tabulaciones; utiliza un ancho de tabulación de 3 ó 4 caracteres en tu editor, para evitar que el código se desplace demasiado a la derecha y tengas que hacer scroll para revisarlo. Sin embargo, si tu editor no permite definir el tamaño de las tabulaciones y el código se desplaza a la derecha, es recomendable que uses espacios en blanco.

Si usas un editor de código que soporte guias de indentación lo verás todo mucho más claro.

Sentencias multilinea

En el caso de sentencias excesivamente largas, es conveniente partirlas en 2 o más lineas. Cuando esto suceda, la linea de continuación debe estar sangrada de manera que quede claro que es una linea de continuación. Como norma general haz el salto de linea después de un operador, tras cerrar un paréntesis o finalizar una condición, de tal manera que la linea de continuación comience por un identificador, paréntesis o nueva expresión. Ejemplo:

```
while (seguir!='s' && seguir!='S' &&
       seguir!='n' && seguir!='N'){
   cin >> seguir;
}
```

Las líneas en blanco entre sentencias son una herramienta muy útil para mejorar la legibilidad del código; si lo crees necesario, no dudes en poner una o dos líneas en blanco para separar sentencias (especialmente al final de los bloques entre llaves).

Operadores y condiciones

A la hora de escribir una condición usando operadores o simplemente una asignación, deja un espacio en blanco antes y después del operador. Si la condición es múltiple y no usas paréntesis, utiliza los espacios en blanco para ganar claridad. Sin embargo, si utilizas demasiados espacios en blanco puedes conseguir el efecto contrario al deseado.

```
// correcto
if (a == 5+34*7){
  ...
```

```
}

while (seguir!='s' && seguir!='S' && seguir!='n' && seguir!='N')
...

// menos correcto
if(a==5){ ...
}
while (seguir!='s'&&seguir!='S'&&seguir!='n'&&seguir!='N')
```

Comentarios

Debes comentar tu código mientras escribes, y hacerlo a varios niveles. Comenta las funciones indicando qué tarea realizan, y aquellos trozos de código que más te haya costado escribir, pues te ayudará a entenderlo cuando lo revises o lo tengas que explicar. Sé correcto en los comentarios, escribe con corrección y no comentes trivialidades, por ejemplo:

```
a = 7;   // asigno un 7 a la variable a (COMENTARIO TRIVIAL, INCORRECTO)
```

Un programa sin comentarios puede ser difícil de entender (excepto cuando el código es muy claro), y por otro lado un programa con demasiados comentarios no deja ver el código y es también muy difícil de entender. El objetivo de los comentarios es facilitar la comprensión del código a otra persona, por lo que es más importante la calidad de los comentarios que la cantidad.

Para los comentarios de varias lineas utiliza la formula

```
/*
** comentario de varias lineas
** otra linea de comentario
*/
```

Los comentarios referidos a una sola línea irán a la derecha de la linea, tras el `;` . Si tienes varias lineas seguidas comentadas, sangra los comentarios al mismo nivel.

```
int cajaf = 3*(fila/3);          // primera fila de la caja
int cajac = 3*(columna/3);       // primera columna de la caja
int caja[SIZE];                  // vector para comprobaciones
```

Sentencias básicas

Escribe las sentencias de acuerdo a tu estilo de sangrado. Nunca escribas las acciones en la misma linea que las condiciones.

Sentencia `if-else`

La estructura básica es

```
if (condición)
   sentencia1;
else
   sentencia2;
```

y en el caso de usar varias sentencias en cada bloque:

```
if (condición){
   sentencia11;
   sentencia12;
   ...
}
else{
   sentencia21;
   sentencia22;
}
```

No es recomendable escribir algo como esto:

```
if (condición) sentencia1;
else sentencia2;
```

En general, si solamente hay una sentencia en el bloque del `if` o en el del `else` no es necesario poner llaves, pero si no se ponen llaves puede haber confusiones peligrosas:

```
if (condición1)
   if (condición2)
      sentencia1;
else
   sentencia2;
```

En este caso, el `else` se corresponde con el `if` más cercano, el de la `condición2`, aunque se encuentre sangrado igual que el otro `if`. Otro caso similar es el siguiente:

```
if (condición)
   sentencia1;
else
   sentencia2;
   sentencia3;
```

Podría parecer por el sangrado que `sentencia3` sólo se ejecuta cuando no se cumple la condición, pero sin embargo se ejecuta siempre, se cumpla o no la condición.

Si es necesario poner una serie de `if-else-if-else-if-else` relacionados entre sí, lo recomendable es ponerlos de esta manera:

```
if (condición1){
   ...
}
else if (condición2){
   ...
}
else if (condición3){
   ...
}
else{
   ...
}
```

Sentencia `while`

La estructura básica es la siguiente:

```
while (condición)
  sentencia;
```

y en el caso de bloque:

```
while (condición){
   sentencia1;
   sentencia2;
   ...
}
```

utiliza siempre un bloque con llaves, aunque solamente tenga una sentencia.

Sentencia `for`

La estructura básica es la siguiente:

```
for (inicialización; condición; incremento)
   sentencia;
```

y en el caso de bloque

```
for (inicialización; condición; incremento){
   sentencia1;
   sentencia2;
   ...
}
```

Para terminar

Programar es una actividad compleja que requiere de mucha práctica, pero cuando se controla es muy gratificante. La valía de un programador no se mide únicamente por sus conocimientos de programación, sino por cómo afronta su

profesión. Intenta aprender de los muchos recursos de programación que puedes encontrar en internet.

Como recomendaciones, te damos los siguientes enlaces:

- Cómo no hacer unas prácticas de programación

  `http://www.di.uniovi.es/~cernuda/pubs/jenui2002-2.pdf`

- 13 consejos para comentar tu código

  `http://www.variablenotfound.com/2007/12/13-consejos-para-comentar-tu-cdigo.html`

- Los principios del programador

  `http://www.developerdotstar.com/mag/articles/PDF/DevDotStar_Read_Principios.pdf`

- Disparar y avanzar

  `http://spanish.joelonsoftware.com/Articles/FireAndMotion.html`

- 20 tips para ser mejor programador

  `http://www.kabytes.com/programacion/20-tips-para-ser-un-mejor-programador/`

- Consejos para estudiantes de informática

  `http://www.joelonsoftware.com/articles/CollegeAdvice.html`

- Cómo hacer lo que amas

  `http://www.fduran.com/wordpress/como-hacer-lo-que-amas/`