Breve Guía
Buenas Prácticas de Programación en C

Ismael Figueroa
ismael.figueroa@ucv.cl

Las *buenas prácticas de programación* son un conjunto formal o informal de reglas, pudiendo ser opcionales u obligatorias, que se adoptan con el fin general de mejorar la *calidad del software*. En particular, se pueden obtener beneficios específicos tales como:

- Facilitar el proceso de desarrollo para el programador.
- Aumentar o mejorar la *legibilidad* y *mantenibilidad* del código fuente, lo que ayuda a otros desarrolladores—o a versiones futuras del autor inicial—a comprender el software.
- Evitar cierta clase de errores comunes, por ejemplo equivocarse por 1 en los límites de una iteración.
- Y muchos otros!

En general, las buenas prácticas existen en las diversas etapas del proceso de desarrollo de software. En este documento nos enfocamos en las buenas prácticas de programación para el lenguaje C. En este contexto, las buenas prácticas nos indican cómo escribir ciertas secciones de nuestros programas, como por ejemplo:

- El largo máximo de cada línea.
- Convenciones de nombres para las variables.
- Cómo indentar el código dentro de las estructuras de control y funciones.
- etc.

Para cada lenguaje de programación existen distintos estándares y conjuntos de buenas prácticas. En particular para el lenguaje C destacamos dos *estilos* principales:[1]

- *El estilo GNU*, fue popularizado por Richard Stallman y es parte de los "GNU Coding Standards" utilizados en el proyecto GNU.

[1] Puede ver una discusión sobre estos y otros estilos en la Wikipedia: `http://en.wikipedia.org/wiki/Indent_style`

- *El estilo K&R*, es el estilo "original" de programación en C, utilizado por Kernighan y Ritchie en el libro "El Lenguaje de Programación C".

En todo caso, sin importar la existencia de múltiples y variados estilos de programación en C, *el principio general es que usted como desarrollador adapte un estilo propio y sea consistente con él!* Para ayudarlo a comenzar, este documento describe una serie de prácticas "obligatorias" que usted puede seguir para obtener una bonificación en sus tareas. La idea es que usted desarrolle su propio estilo de programación, que sea claro y conciso, aunque puede no corresponder 100 % con las indicaciones de este documento.

En los casos a continuación que presentan ejemplos, el código a la izquierda (o arriba) **no sigue la buena práctica**, mientras que el código a la derecha (o abajo) sí lo hace. No dude en escribirme con comentarios y nuevos contenidos para este documento—que es un constante trabajo en progreso :)!

Sobre la estructura del programa

1. Utilice indentación para todas las operaciones que se realizan dentro de una estructura de control, tal como un `if`, `for`, `while` o `do-while`, y para el contenido de funciones, registros y otros constructos del lenguaje. Esto resalta la estructura lógica del código y simplifica su lectura. Por ejemplo:

```
while (n != 0) {
d = n%10;
n = n/10;
if(d > 5) {
printf("%d", d);
}
}
```

```
while (n != 0) {
  d = n%10;
  n = n/10;
  if(d > 5) {
    printf("%d", d);
  }
}
```

2. Siempre utilice llaves en todas las estructuras de control—incluso si tienen sólo 1 instrucción en su interior!. Esto le ayudará a evitar problemas si posteriormente desea agregar más instrucciones—típicamente `printf`—bajo dicha estructura de control. La única excepción aceptable a esta regla es poner la instrucción a ejecutar en la misma línea que la instrucción de control. Ejemplo:

```
for(i = 0; i < n; i += 1)
  if(i % k == 0)
    suma += arreglo[i];
```

```
for(i = 0; i < n; i += 1) {
  if(i % k == 0) {
    suma += arreglo[i];
  }
}
```

```
for(i = 0; i < n; i += 1)
    s += a[i];
```

```
for(i = 0; i < n; i += 1) s += a[i];
```

3. Utilice consistentemente un estilo de apertura y cierre de llaves. Por ejemplo:

```
if (a > 10) {
  printf("lol");
}
```

```
if (a > 10)
{
  printf("lol");
}
```

```
if (a > 10)
  {
    printf("lol");
  }
```

   Observe que en la primera opción (de izquierda a derecha) siempre debe poner un espacio antes de la llave inicial `{`. Además, no escriba código justo después de una llave de apertura `{`. Por ejemplo:

```
if (a > 10) { printf("lol");
  printf("wow");
}
```

4. Utilice sólo una instrucción por línea. Por ejemplo:

```
while (n != 0) {
  d = n%10; n = n/10;
  if(d > 5) {
    printf("%d", d); suma += d;
  }
}
```

```
while (n != 0) {
  d = n%10;
  n = n/10;
  if(d > 5) {
    printf("%d", d);
    suma += d;
  }
}
```

5. Utilice un espacio después de las comas. Por ejemplo:

```
int x,y,suma,digitos,factorial;
if (x < pow(y,suma)) {
  printf("%d %d %d",x,y,z);
}
```

```
int x, y, suma, digitos, factorial;
if (x < pow(y, suma)) {
  printf("%d %d %d", x, y, z);
}
```

6. Utilice paréntesis para especificar la estructura de operaciones aritméticas complejas, en vez de esperar que el lector del código se recuerde de todas las reglas de precedencia de los operadores.

7. Utilice espacios a ambos lados de los operadores binarios. Así, se resalta el operador y se facilita la lectura del programa. Por ejemplo:

```
if(x==y&&z%w==0) {
  printf("%d", x+y/z*w);
}
```

```
if(x == y && z % w == 0) {
  printf("%d", (x + y) / (z * w)));
}
```

   La única excepción aceptable es para los índices de los arreglos, por ejemplo: `numeros[i+1]`.

8. Elija entre utilizar o no utilizar espacios antes de cada llamada a función, y cada utilización de una estructura de control. Por ejemplo:

```
for(i = 0; i < n; i += 1) {
  if(fib(n) % 5 == 2) {
    suma += max(n, fib(n-5));
  }
}
```

```
for (i = 0; i < n; i += 1) {
  if (fib (n) % 5 == 2) {
    suma += max (n, fib (n-5));
  }
}
```

9. El programa no debe tener instrucciones *inalcanzables*, es decir, que no es posible que se ejecuten bajo ningún parámetro o dato de entrada. Un ejemplo de instrucción inalcanzable:

```
if (n > 0) {
  while (n <= 0) {
    printf("nunca se imprime");
  }
}
```

10. En una estructura de control `for` utilice las secciones de inicialización e incremento sólo para acciones relacionadas con la variable de control. Por ejemplo:

```
/* mala practica */
for(i = 0, suma = 0; i < n; suma += numeros[i], i += 1);

/* mejor escribirlo asi */
suma = 0;
for(i = 0; i < n; i += 1) {
  suma += numeros[i];
}
```

11. Trate de evitar el uso de los operadores `++` y `-`. Dependiendo de su uso, pueden ocurrir errores sutiles. Por ejemplo:

```
int i = 1, j;
j = i++;

/* i = 2, j = 1 */
printf("i: %d j: %d", i, j);
```

```
int i = 1, j;
i += 1;
j = 1;

/* i = 2, j = 2 */
printf("i: %d j: %d", i, j);
```

12. Siempre utilice `break` para terminar una iteración, en vez de modificar la variable de iteración. El código que sigue luego del ciclo puede depender del valor de dicha variable.

13. Si utiliza la instrucción `switch`, siempre agregue un caso `default`. Este caso debe ser el último caso.

Sobre las variables

1. No utilice variables globales.
2. Utilice nombres significativos para sus variables. La idea es que la lectura del código sea "auto-explicativa", lo que facilita su posterior modificación y comprensión. Por ejemplo:

```c
int x, f, z
scanf("%d", &z);
f = z;
while(f != 0) {
  x += f % 10;
  x = x / 10;
}
```

```c
int n, temp, sumaDigitos;
scanf("%d", &n);
temp = z;
while(temp != 0) {
  sumaDigitos += temp % 10;
  temp = temp / 10;
}
```

   Observe que hay convenciones comunes, conocidas como *idioms*, específicos de cada lenguaje. Por ejemplo:

   - Las variables a ser usadas como índices para iteración suelen comenzar desde `i`. O sea, `i`, `j`, etc.
   - En el curso se suele usar `n` como algún dato ingresado por el usuario y que es un parámetro del problema. Otros parámetros son desde `m` en adelante.

3. Escoja una opción entre `camelCase` y `snake_case` para los nombres de variables que tienen más de una palabra. Sea consistente con dicha elección.
4. Todo valor constante debe declararse al comienzo del programa usando `#define`. El nombre de dicha constante debe ser en mayúsculas.
5. Todas las variables deben ser *inicializadas* con algún valor. Por ahora la única excepción aceptable es la inicialización de arreglos y matrices.
6. Puede declarar múltiples variables del mismo tipo (que no sean arreglos o matrices) en una misma línea o en una línea aparte para cada una.
7. Evite las conversiones implícitas entre tipos de dato. Por ejemplo:

```c
int a = 10;
double b;
b = a; /* int a double! */
```

```c
int a = 10;
double b;
b = (double)a;
```

8. Declare cada arreglo o matriz en una línea aparte.
9. Declare cada puntero en una línea aparte.
10. Todo puntero debe ser inicializado a `NULL` en su declaración.

Sobre los comentarios

1. Cada programa debe comenzar con un comentario que describa su propósito, los supuestos realizados, y un ejemplo mínimo con la entrada y salida esperados.

2. Cada bloque de código debe ser precedido por un comentario que explica la función de ese bloque, su propósito y los supuestos que ahí se utilizan.

3. La regla de oro para los comentarios es que *los comentarios deben ser consistentes con el código*. Es muy probable que el programa sea incorrecto si el código y los comentarios no coinciden. Debe ser su prioridad el actualizar los comentarios cada vez que cambia el código fuente.

4. Los comentarios deben ser frases completas, donde su primera palabra comienza con mayúsculas, a menos que sea un nombre de variable que comience con minúsculas, y que terminan con un punto. Por ejemplo:

```
/* recorrer el arreglo y chequear
 * que todos los elementos sean cero */

/* Recorrer el arreglo y chequear
 * que todos los elementos sean cero.
 */
```

   La única excepción es para comentarios muy cortos, como por ejemplo:

```
i = 1; /* indice de iteracion */
```

5. Utilice comillas simples para referirse a alguna variable. Por ejemplo:

```
/* La variable 'n' representa la cantidad de elementos en el arreglo */
```

6. Los comentarios deben ser correctos gramaticalmente y ortográficamente. Como excepción, pueden omitirse los acentos ortográficos.

7. Ponga un espacio después de la apertura del comentario, y antes del cierre del mismo. Por ejemplo:

```
/*Este comentario es mas dificil de leer*/


/* Este comentario es mas facil de leer */
```

8. Utilice el siguiente formato para comentarios con múltiples líneas:

```
/*
 * Este es un comentario,
 * con multiples lineas.
 */
```

9. Los comentarios de bloque se refieren al código que los sigue a continuación, y deben tener el mismo nivel de indentación que dicho código. Los párrafos dentro de un comentario de bloque se separan con una línea en blanco. Por ejemplo:

```
/*
 * La primera linea viene despues de una linea vacia.
 *
 * Los otros parrafos van separados por una linea vacia
 * y tambien puede haber una linea vacia al final (opcionalmente).
 *
 */
```

10. Escriba comentarios para todo lo que no sea completamente obvio al leer el código fuente. En particular, escriba comentarios para cualquier "truco" que esté usando. En caso de duda, opte por escribir más comentarios y no menos!

11. Evite escribir comentarios totalmente redundantes, como por ejemplo:

```
i = 1; /* asignar 'i' en 1 */
```

Y evite además comentarios que no tienen sentido, como por ejemplo:

```
/* i */
i = 1;
```