Programación

Guía de Estilo

1. Introducción

Este documento no pretende ser un decálogo de cómo programar, tan sólo marcar unas pautas comunes a todos los involucrados (profesores y alumnos) en la asignatura.

Esta guía de estilo no sólo abarca cuestiones relativas al aspecto del código (tabulado, identificadores suficientemente explicativos, etc.), sino también aspectos del diseño del programa y uso adecuado de las estructuras de control (alternativas y repetitivas) y de los tipos de datos estructurados (arrays y registros).

El código de un programa lo leerán muchas personas, incluido el mismo autor inicial, para corregirlo, ampliarlo o simplemente evaluarlo. Para todos ellos es fundamental que esté bien redactado, que su significado sea claro y que esté correctamente estructurado.

Tener o no estilo, puede afectar dramáticamente a la calidad de un programa. Un código de calidad debe estar escrito de tal manera que sea correcto, eficiente, de fácil lectura y de fácil actualización. Elegir y seguir un determinado estilo puede incrementar la probabilidad de conseguir estos cuatro objetivos.

2. Normas de Estilo

Redactar el código con estilo puede considerarse más un arte que una ciencia, pero existen una serie de normas que ayudan a conseguir un código fuente satisfactorio. Este es el objetivo de esta guía. Algunas de estas normas y recomendaciones pueden variar al emplear otro lenguaje de programación, pero el objetivo será siempre el mismo: conseguir un código satisfactorio.

En cualquier caso, un programa debe cumplir los requisitos de un algoritmo (preciso, unívoco, finito), además de ser eficaz, legible y sencillo de modificar/ampliar. Y por último, como requisito deseable, pero no exigible, eficiente.

Nomenclatura de los identificadores.

- Los identificadores de constantes, variables, registros y funciones deben ser consecuentes con su semántica. Es decir, el nombre de los identificadores debe explicar el valor que almacenan, el tipo de información que definen, o la tarea que realizan. Para conseguir este objetivo, los identificadores de variables, constantes y estructuras suelen ser sustantivos mientras que los identificadores de funciones verbos activos.
- No se deben utilizar nombres muy largos, ni tampoco tan cortos que se pierda su significado. En cualquier caso se dará preferencia a la claridad frente a la brevedad.
- C es un lenguaje que distingue entre mayúsculas y minúsculas. A la hora de establecer el identificador de constantes, variables, registros o funciones, se recomienda seguir estas normas:
  - ✓ Los identificadores de las variables, registros y funciones se escribirán en minúsculas pero si el nombre consta de varias palabras la inicial de la segunda y subsiguientes se escribirán en mayúsculas. Si la variable es de tipo puntero su nombre empezará por "*ptr*".

Ejemplos:
```
int dia, numAlumnos;
float notaMedia;
int *ptrVar;      //Puntero a una variable
int guardarEstadisticas (void);
struct punto{
     float x;
     float y;
  }
```

✓ Los identificadores de las constantes siempre en mayúsculas. Si el nombre consta de varias palabras se escribirán separadas por un guión bajo.

Ejenplo:
```
#define MAX_FILAS 20
```

Uso de variables.

- Las variables deben declararse en su ámbito de uso y siempre al inicio de la función para la que son visibles. **Nunca se deben usar variables globales.**
- Las variables para el control de iteraciones (de tipo entero) tendrán nombres cortos como `i`,`j`,ó `k`
- Se debe evitar el uso de una misma variable para almacenar dos o más cosas distintas.
- Las variables de un mismo tipo se pueden declarar en una misma línea de código pero sólo se recomienda alinear aquellas variables que tienen un significado similar.

  Ejemplo:
  ```
  int i, j, k; //Variables de control
  int num;     //Almacena un num introducido por teclado.
  ```
- Siempre deben estar inicializadas. Además es aconsejable que la inicialización se realice junto a la declaración.

  Ejemplo:
  ```
  int contador=0;.
  ```

Expresiones.

- Evitar expresiones numéricas o condiciones complejas. Dividir en subexpresiones.
- En una estructura condicional, *if* llevará la condición que se cumple más comúnmente. La menos común será para el *else*.
- Aclarar el orden de las operaciones involucradas en una expresión con los paréntesis correspondientes.

Estructura de un Programa en C

- Aunque son permisibles otras opciones, el programa principal (función *main*) debería ajustarse a la siguiente sintaxis:

```
int main (void){
  …
  return 0;
}
```

- El programa principal, siempre que sea posible, no deberá ocupar más de una pantalla (20-30 líneas de código sin incluir comentarios) ni tampoco debe constar de 1, 2, ó 3 líneas de código.
- En las estructuras de control condicionales se deben tener presentes todas las posibilidades de la expresión que determina la selección. Es decir, el programador debe haber previsto las situaciones posibles, pero no tiene necesariamente que incluir todas las ramas de la estructura condicional.
- La elección de un *while*, *do-while* o un *for* se debe corresponder a sus diferencias (número de iteraciones fijo o no).
- En un *for* nunca se debe modificar ni la variable índice ni el límite superior.
- Un bucle (incluido su "cuerpo") no debería ocupar más de una pantalla (20 líneas de código aprox.). En otro caso, se deben emplear funciones para reducir el tamaño del cuerpo del bucle.
- Evitar el anidamiento excesivo de estructuras condicionales (*if*, *switch*) y/o iterativas (*for*, *while*, *do-while*). Si hay más de estas tres estructuras anidadas emplear funciones para reducir el número de ellas.
- Un conjunto de instrucciones no trivial que se repite a lo largo del código, ha de ser reemplazado por la correspondiente función.

Funciones.

- Debe evitarse el diseño de funciones demasiado largas. En el caso de los subprogramas, más de dos pantallas de código se puede considerar como demasiado largo.
- Los parámetros de una función representan datos de entrada, de salida o de entrada-salida.
  - ✓ Los datos de entrada deben pasarse a la función por valor.
  - ✓ Los datos de entrada-salida deben pasarse a la función por referencia.
  - ✓ Si la función sólo tiene un dato de salida, éste será el valor de retorno de la función.
  - ✓ Si la función debe modificar dos o más datos se debe emplear obligatoriamente el paso por referencia. En este caso se recomienda crear una función tipo *void* y pasar todos los parámetros de salida por referencia.

- En una función todas las variables deben declararse como parámetros o como variables locales.
- Si una función no acepta parámetros debe indicarse con la palabra reservada *void.*

  Ejemplo: `int mostrarMenu(void);`
- Si una función no devuelve ningún valor, debe declararse como tipo *void*.

  Ejemplo: `void mostrarDatos(int a, int b);`
- En una función, que tiene que devolver un resultado, se recomienda incluir siempre una única sentencia *return* y que ésta esté situada al final de la función.
- A la hora de implementar una función se recomienda definirla haciendo uso del prototipo. Además, este prototipo debería incluir el nombre (y no solo el tipo) de los parámetros que recibe.

Estructuras (Registros).

- Para evitar errores de compilación, las estructuras deben definirse antes que los prototipos de las funciones que las usan. Por tanto, se recomienda declarar todas las estructuras tras las directivas `#include`.
- Cada campo de una estructura debe declararse en un renglón separado.

  Ejemplo:
```
struct tipoLibro{
  char titulo [TAM]
  char autor[TAM];
  int edicion;
}
```
- Los campos de un registro deben reflejar de forma natural las entidades que lo componen. Se recomienda por tanto no usar construcciones de tipos denominados "anónimos". Por el contrario se recomienda declarar las componentes e identificarlas una a una.

  Ejemplo:
```
struct tipoAutor{
  char nombre[TAM];
  char apellido[TAM];
};

struct tipoLibro{
  char titulo [TAM]
  tipoAutor autor;
  int edicion;
};
```

Vectores.

- El tamaño de un vector debe estar declarado como una constante.
  Ejemplo:
```
#define TAM 20
int main(void){
  int vector[TAM];
  …
  return 0;
}
```
- Cuando un vector se pasa como parámetro a una función es buena práctica pasar el tamaño del vector como parámetro adicional. Con esto se consigue dar mayor generalidad a las funciones.

Edición.

- No escribir líneas de más de 70 caracteres. En caso contrario, partir la línea ya sea comentario o instrucción de código.
- Emplear dos/tres espacios como medida básica de sangrado, no emplear el carácter tabulador (#09 de la tabla ASCII)
- En estructuras de bloque (condicionales o iterativas) con una única sentencia se pueden obviar las llaves que delimitan el bloque de instrucciones Sin embargo, al ampliar el bloque añadiendo instrucciones el no tener marcado el inicio y el final suele ser una fuente de error.
- Las instrucciones de un mismo bloque deben ir todas ellas con el mismo sangrado. El cuerpo de una estructura condicional o iterativa se tabulará con dos espacios a la derecha con respecto a la línea "cabecera" de la instrucción

```
int main (void){
  instrucción_1;
  instrucción_2;
  ...
  if (expresión){
    instrucción_if_1;
    instrucción_if_2;
    …….
  } //fin if
  ...
  return 0;
};
```

- Se recomienda incluir la llave, "*{*", que marca el inicio de un bloque de código a la derecha de la sentencia que introduce el bloque. La llave, "*}*", que marca el final del bloque debe colocarse en la misma columna que dicha sentencia (igual tabulado) pero en una línea en la que no exista código ejecutable. Además resulta muy aconsejable marcar con comentarios el final de cada bloque (Ver ejemplo anterior).
- Se aconseja no utilizar más de una instrucción por línea de código aunque es permisible agrupar en una misma línea sentencias similares.

  Ejemplo:

```
printf ("Introduzca un número "); scanf ("%d", &n);
```

Comentarios.

- Todo programa debería encabezarse con los datos que permitan identificar al autor (Nombre, Apellidos, NIA y Grupo) y con un breve resumen de los objetivos que ha de satisfacer.

  Ejemplo:

```
/*****************************************************************
*    Programa: factorial.c                                       *
*    Objetivo:  calcular el factorial de un numero               *
*    Version: 1.0 del 10 de julio de 2013                        *
*    Autor:                                                      *
*    Grupo:                                                      *
*    NIA:                                                         *
 *****************************************************************/
```

- Toda función debería comenzar indicando el objetivo que persigue y el significado de sus parámetros..
- Se aconseja que, en el curso de la edición de un programa, se marquen con una fecha las modificaciones que se han realizado en una función.

  Ejemplo:
```
void mostrarDatos(int a, int b){
    /*Modificada el 20-5-2012*/  ..
      return;
    };
```

- Se deben añadir explicaciones a todo lo que no es evidente pero también hay que evitar las redundancias. Es decir, no hay que repetir lo que se hace, sino explicar, de forma clara y concisa, por qué se hace.