Consejos de Estilo de Programación en C

1 Estructura del programa

La estructura básica de un programa de un sólo módulo está compuesta por una cabecera, instrucciones `#include`, instrucciones `#define`, instrucciones `typedef`, prototipos, la función `main`, y el resto de las funciones. (ver ejemplo al final). Con este orden se garantiza la coherencia de todas las definiciones, mientras que utilizar un orden diferente sólo sería válido en algunas situaciones particulares.

Crear una cabecera de comentarios para documentar cada módulo y cada función.

2 Variables

Escribir los nombres de variables en minúsculas

Si el nombre es compuesto, utilizar el guión bajo para separar palabras. Por ejemplo: `suma_complejos`

Es conveniente que el nombre de las variables tenga sentido, para ayudar a entender el programa. Es habitual utilizar algún criterio para identificar el tipo de variable por su nombre, por ejemplo que todos los punteros a entero utilicen nombres de variables empezando por `pi_`

Es muy recomendable añadir un comentario al lado de la declaración para explicar mejor su significado. Declarar muchas variables es una sola línea impide escribir el comentario.

Evitar utilizar variables globales. Sólo en casos muy especiales tiene sentido utilizar variables globales; en esos casos hay que documentar en cada función qué variables globales se utilizan.

Es recomendable inicializar variables justo delante del bucle que requiere la inicialización. Esto facilita la revisión del programa, evita que la inicialización se pueda perder antes de llegar al bucle, y facilita la reutilización de fragmentos de código. En cualquier caso el código es más claro si la inicialización se realiza en una línea diferente a la declaración de la variable.

No inicializar variables que no requieran inicialización. Esto anula la capacidad del compilador de detectar el uso de una variable antes de haberle dado valor (por ejemplo cuando uno se confunde de nombre de variable o olvida en signo & en un `scanf`).

3 Funciones

Escribir los nombres de las funciones con la primera letra mayúscula y las siguientes minúsculas.

Si el nombre es compuesto, utilizar mezclas de minúsculas y mayúsculas. Por ejemplo: `SumarComplejos()`. Resulta más claro utilizar nombres de acción para nombres de funciones.

Cada función debe llevar su propia cabecera de descripción, que es equivalente al manual de referencia.

Evitar escribir funciones muy largas. Es conveniente dividir una función larga en varias funciones pequeñas para facilitar la comprensión del programa y la depuración, aunque dichas funciones sólo se llamen una vez.

4 Claridad del Código

Seguir las recomendaciones de estilo de nombres de parámetros y tipos (mayúsculas), nombres de variables (minúsculas) y nombres de funciones (mezcla).

En general utilizar nombres en español para objetos declarados por el programador. Esto permite diferenciarlos de los objetos propios del lenguaje que son nombres en inglés.

Utilizar un correcto sangrado del texto para localizar rápidamente el inicio y el final de cada bloque de código.

No escribir líneas de código demasiado largas. En C todas las expresiones matemáticas, expresiones lógicas y llamadas a función se pueden separar en varias líneas. Las cadenas de caracteres se pueden separar en varias líneas cerrando las dobles comillas en una líneas y abriéndolas en la siguiente.

No utilizar operaciones crípticas, porque puede no quedar clara la instrucción. Por ejemplo:

```
if ((b[i]=a[++i])==0) ...
```

Por otro lado, algunas de estas operaciones son ambiguas:

`a = ++c + c;` es dependiente del compilador (A. Kelly, I. Pohl, "A book on C").

Algunas operaciones está bien definidas en el lenguaje, pero no resultan intuitivas. Por ejemplo: `*p++` no es igual a `(*p)++`

Escribir instrucciones complejas en varias líneas y utilizar paréntesis evita ambigüedades y ayuda a entender el código.

Es más seguro que las llamadas a `calloc` y `free`, o las llamadas a `fopen` y `fclose` aparezcan dentro de la misma función. Asignar memoria en una función y liberar en otra supone mayor riesgo de no hacer las cosas bien. Son una excepción a esta regla las estructuras especiales de datos como listas y árboles, o las funciones específicas para asignación de memoria.

En programas con interfaz gráfica es fundamental separar las funciones de control de la interfaz de las funciones de cálculo. Esto permite depurar las funciones de cálculo de manera independiente y con programas auxiliares, además de facilitar la portabilidad si se decide cambiar de herramienta gráfica.

Debe utilizarse la estructura de bucle adecuada para cada caso: `for, while` ó `do-while.` Como norma general el bucle `for` se utiliza cuando el número de iteraciones es fijo y conocido desde el principio. Por lo tanto la condición de salida del bucle `for` no debería ser muy compleja, ni tiene sentido modificar el contador dentro del bucle.

Las salidas a mitad de bucle con `break` o `continue` están totalmente desaconsejadas. Estas instrucciones se pueden evitar utilizando bloques `if` o añadiendo variables de control de flujo en los bucles `while` y `do-while`

Está desaconsejada la utilización de `exit` en cualquier parte del código y la utilización de `return` en medio de una función. Salvo casos excepcionales, resulta más claro que las funciones tengan un único punto de entrada y un único punto de salida.

5 Documentación

En apuntes, trabajos, enunciados de prácticas y exámenes, facilita la comprensión utilizar tipos monoespaciados para escribir nombre de funciones como `printf`.

Los listado de programas deben imprimirse utilizando un tipo de letra monoespaciado (como por ejemplo `Courier` o también `Lucida Console`) y un tamaño adecuado. La conversión de tabuladores a espacios evita que se descoloque el texto.

6 Ejemplo

```
/*
PROGRAMA: Ejemplo de Estilo.   Programa para calcular la suma de un conjuntos de
complejos
FECHA: 5/may/2003
Autor: Rafael Palacios
*/

#include <stdio.h>
#define N 10
typedef struct {
    double real;   /* Parte Real*/
    double imag;   /* Parte Imaginaria */
} COMPLEJO;

COMPLEJO SumarComplejos(COMPLEJO a[], int n);

void main(void)
{
    COMPLEJO v[N]; /* vector de complejos */
    int n;    /* número de elementos */
    int i;    /* contador */
    COMPLEJO suma; /* suma de los complejos */

    /*** Pido el número de elementos ***/
    do {
        printf("Cuantos elementos quieres introducir? ");
        scanf("%d",&n);
        if (n>N)  {
            printf("Este programa no soporta tantos elementos\n");
        }
    } while (n>N);

    /*** Pido los n elementos ***/
    for(i=0; i<n; i++)  {
        printf("Parte real del elemento %d: ",i+1);
        scanf("%lf",&v[i].real);
        printf("Parte imaginaria del elemento %d: ",i+1);
        scanf("%lf",&v[i].imag);
    }

    /*** Calcula la suma de todos los complejos ***/
    suma=SumarComplejos(v,n);

    /*** Salida de resultados ***/
    printf("La suma es: (%lf,%lf)\n",suma.real, suma.imag);

}

/*************** F U N C I O N E S ***************************************/
/*
Función SumarComplejos
Descripción: Calcula la suma de un vector de complejos
Argumentos:
   a  vector de complejos
   n  número de elementos válidos del vector
Valor retornado:
   suma complejo resultados
Advertencias:
   En caso de error, por ejemplo el número de elementos es menor que 1,
   la función devuelve el complejo (0,0);
*/
COMPLEJO SumarComplejos(COMPLEJO a[], int n)
{
    COMPLEJO ret;  /* valor a retornar */
    int i;         /* contador */

    ret.real=0;
    ret.imag=0;
    for(i=0; i<n; i++)  {
        ret.real += a[i].real;
        ret.imag += a[i].imag;
    }

    return ret;
}
```