Estructura del programa

La estructura básica de un programa de un sólo módulo está compuesta por una cabecera, directivas #include, directivas #define, typedef, prototipos, y la función main. Con este orden se garantiza la coherencia de todas las definiciones, mientras que utilizar un orden diferente sólo sería válido en algunas situaciones particulares.

```
/*****************************************************************
* Programa: Ejemplo Clase X-
*
* Objetivo:
   Ingresar datos de 1 persona e imprimirlos.
       nombre [32]
       edad
*
* Version: 0.1 del 06 enero de 2016
* Autor: Ernesto Gigliotti
*
*****************************************************************/

#include <stdio.h>
#include <stdlib.h>

struct S_Person
{
      char name[20];
      int age;
};
typedef struct S_Person Person;

int enterPerson(Person* p);
void printPerson(Person* p);

int main(int argc, char *argv[])
{
…....
}
```

Variables

1. Escribir los nombres de variables en minúsculas y si el nombre es compuesto, utilizar mayuscula en la primer letra de la nueva palabra, este estilo lleva el nombre de “lowerCamelCase”.

```
auxiliarInt
```

2. Es conveniente que el nombre de las variables tenga sentido, para ayudar a entender el programa y que el mismo se encuentre en inglés.

3. 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 con “p” por ejemplo “**pCounter”** .

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

5. 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.

6. 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.

Ejemplo:

```
int i=9;
while(i>0)
{
   ...
```

7. Las variables para el control de iteraciones (de tipo entero) podrán tener nombres cortos como i , j , ó k.

Funciones

1. Escribir los nombres de las funciones con la primera letra minúscula y si el nombre es compuesto, utilizar mayúscula en la primer letra de la nueva palabras

```
getFloat(char mensaje[])
```

2. Resulta más claro utilizar nombres de acción para nombres de funciones.

3. Utilizar nombres en Inglés para las funciones, esto permitirá obtener un código que podrá ser interpretado por cualquier programador en el mundo.

4. Evitar escribir funciones con mucho código. 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.

5. Cada función debe llevar su propia cabecera de descripción, que es equivalente al manual de referencia (utilizar el estilo Doxygen). Doxygen es un generador de documentación para C++, C, Java, Objective-C, Python, IDL (versiones Corba y Microsoft), VHDL y en cierta medida para PHP, C# y D. Dado que es fácilmente adaptable, funciona en la mayoría de sistemas Unix así como en Windows y Mac OS X.

```
/**
* \brief Solicita un número al usuario y devuelve el resultado
* \param mensaje Es el mensaje a ser mostrado
* \return El número ingresado por el usuario
*
*/
float getFloat(char mensaje[])
{
   float auxiliar;
   printf("%s",mensaje);
   scanf("%f",&auxiliar);
   return auxiliar;
}
```

6. 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 misma

7. 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.

8. En una función todas las variables deben declararse como parámetros o como variables locales.

9. Si una función no acepta parámetros debe indicarse con la palabra reservada void.

```
int mostrarMenu(void);
```

10. Si una función no devuelve ningún valor, debe declararse como tipo void.

```
void mostrarDatos(int a, int b);
```

11. 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. Queda excluida de esta regla el caso de las funciones de validación que retornan 0 o 1.

12. 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.

Vectores.

1. El tamaño de un vector debe estar declarado como una constante.

```
#define TAM 20
int main(void)
{
	int vector[TAM];
	…
	return 0;
}
```

2. 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.

Claridad del Código

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

```
for(i=0;i<4;i++)
{
    printf("Ingrese un numero: ");
    scanf("%d",&auxiliarInt);
    acumulador = acumulador + auxiliarInt;
    if(auxiliarInt > maximo)
    {
        maximo = auxiliarInt;
    }

}
```

2. 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. Se recomienda SIEMPRE utilizar llaves.

3. No escribir líneas de código demasiado largas, como máximo 80 caracteres. En C todas las expresiones matemáticas, expresiones lógicas y llamadas a función se pueden separar en varias líneas.

4. El programa principal, siempre que sea posible, no deberá ocupar más de una pantalla (40 líneas de código sin incluir comentarios) .

5. Las cadenas de caracteres se pueden separar en varias líneas cerrando las dobles comillas en una línea y abriéndolas en la siguiente.

6. 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.

7. 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.

8. 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.

9. Un conjunto de instrucciones no trivial que se repite a lo largo del código, ha de ser reemplazado por la correspondiente función.

10. La directiva #define es una palabra clave del pre-procesador su nombre siempre irá en mayúsculas. Si el nombre consta de varias palabras se escribirán separadas por un guión bajo.

```
#define MAX_FILAS 20
```

11. Se aconseja no utilizar más de una instrucción por línea de código

12. 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.