Buenas prácticas de Programación

1. Escribe tus programas lo más simple y directo posible. *Keep it simple*.
2. Si trabajas con un compilador, ajusta sus opciones para que arroje la mayor cantidad de errores y advertencias posibles al compilar, de ese modo, tu aplicación tendrá menores oportunidades de obtener errores aleatorios. En base a lo anterior, revisa cada mensaje para tomar las medidas pertinentes.
3. Todo programa debe ser previamente comentado, explicando el propósito, funcionamiento completo y el resultado esperado.
4. Dentro de las funciones definidas, establece un espaciado o **indentación**, que resalte la estructura funcional de la aplicación y facilite la lectura al programador al que le corresponda analizar el código.
5. Por lo general, se usa un nivel de indentación por cada bloque de código (sentencias condicionales y bucles son consideradas como bloques de código embebido dentro de otro, por lo que se recomienda la indentación), ésta indentación corresponde a una sangría que comúnmente tiene el valor de una tabulación (tecla Tab) o bien tres o cuatro espacios.
6. Es importante que el tamaño de las sangrías sean regulares (consistentes) y no varíen a lo largo del código, es decir, si el primer bloque ocupa como indentación una tabulación, el resto de bloques deben ser indentados con una tabulación adicional por cada nivel, con eso se facilita la lectura en cualquier editor de código.
7. Se recomienda declarar variables en líneas separadas, ya que se facilita la descripción de cada variable mediante comentarios.
8. Poner un espacio después de cada coma (,) facilita la legibilidad del código.
9. No uses variables cuyo nombre no posea algún significado descriptivo. Una variable con nombres significativos permite al lector entender el contexto del código y permite disminuir la cantidad de documentación asociada, puesto que con un código legible y nombres significativos, el código se ve *auto documentado*. Por ejemplo, una variable llamada *cantidad_recursos*, tiene más significado que una variable llamada *cr*.
10. Se consistente al momento de utilizar un estándar para nombres largos. Puedes usar el estándar usado en C ("nombre_largo"), o bien el utilizado en Java, llamado *CamelCase*("nombre","VariableNombreLargo","Clase","ClaseNombreLargo").
11. Evita el código *commented-out*, que corresponde al código comentado para que no se ejecute/no compile, ya que la lectura del código se vuelve engorrosa.
12. Comenta cuando sea justo y necesario, usa los comentarios dentro de las funciones para describir las variables (sólo cuando su utilidad sea potencialmente dudosa) y cuando existan bloques de código difíciles de entender a primera vista; el exceso de comentarios vuelve ilegible el código.

13. Se recomienda como buena costumbre, añadir al inicio de cada función, un bloque de comentarios que expliquen el comportamiento general de la función, de modo que se pueda entender a grosso-modo que es lo que hace, o se espera que haga; así se facilita la búsqueda de errores, y se evita el análisis innecesario en una gran cantidad de casos.
14. Es **altamente recomendada** la definición de variables locales al inicio de la implementación de cada función, como un bloque de código bien separado del bloque que contenga las instrucciones ejecutables. Ésta separación puede consistir en una línea en blanco, o bien un comentario que denote la utilidad de cada bloque.
15. En caso de usar operadores binarios (por ejemplo +, -, &&, ||, entre otros) se recomienda poner espacio a los extremos de cada operador, de modo que se resalte su presencia y se facilite la lectura del código.
16. Se recomienda en algunas operaciones complejas, hacer uso de paréntesis redundantes o innecesarios que sirven para poder agrupar expresiones dentro de tales operaciones.
17. **Evita la incorporación de más de una instrucción por línea.** Esto reduce notoriamente la legibilidad del código, ya que el programador habitualmente está acostumbrado a leer una instrucción por línea.
18. Si el código soporta la separación de sentencias en varias lineas, procura realizar una separación coherente, en el que cada punto de ruptura tenga sentido.
19. Si una instrucción abarca más de una linea, recuerda realizar la indentación necesaria.
20. Cuando escribas operaciones que hagan uso de muchos operadores, procura revisar que las operaciones se estén realizando en el orden que tu esperas que se realicen, muchas veces el lenguaje tiene otra forma de asimilar la precedencia, por lo que el resultado real varía con respecto al esperado, en general, se recomienda forzar la precedencia de operaciones haciendo uso de paréntesis.
21. Si el lenguaje soporta llaves({}) para la separación de bloques, es altamente recomendado usarlas, ello facilita el proceso de distinción de bloques de código en forma rápida, permitiendo identificar y reparar errores en el código con menos dificultad.
22. Si deseas evitar omitir una llave, abre y cierra el bloque de código que deseas crear, y luego introduce código dentro del bloque, con eso te aseguras la victoria.
23. **Nunca** olvides inicializar los contadores y sumadores (variables de control).