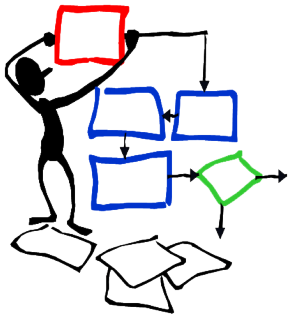
Instituto Politécnico Nacional

Escuela Superior de Cómputo



Algoritmia y programación estructurada

Tema 22: Documentación y reutilización de código

M. en C. Edgardo Adrián Franco Martínez
http://www.eafranco.com
edfrancom@ipn.mx
@edfrancom edgardoadrianfrancom

Contenido

- Software
  - Ingeniería del software
  - Ciclo de vida del software
- Documentación de software
  - Documentación externa
  - Documentación interna
- Código autodocumentado
- Comentarios efectivos
- Técnicas para comentar código
- Reutilización de código
  - Tipos de reutilización
    - Copiar y pegar
- Reutilizar código en C
- Concepto de Librería en Programación
- Biblioteca estándar de C
- Creación de bibliotecas para C
  - Generación de código ejecutable
  - Generando una librería para C

Software

- Comúnmente se asocia el termino *Software* se asocia con *Programa de computadora*. Sin embrago una definición más adecuada dentro del contexto de la ingeniería en sistemas computacionales sería:

Programa de computadora y la documentación asociada a este. [Ian Sommerville, 2005]

- Nota: Los productos de software se pueden desarrollarse para algún cliente en particular o para un mercado general.

Ingeniería del software

- La ingeniería del software es un disciplina de ingeniería que comprende todos los aspectos de la producción de software.

- **¿Cuál es la diferencia entre ingeniería de software y ciencia de la computación?**
  - La ciencia de la computación comprende la teoría y los fundamentos; la ingeniería de software comprende las formas practicas para desarrollar y entregar un software útil.

- **¿Cuál es la diferencia entre ingeniería de software e ingeniería de sistemas?**
  - La ingeniería de sistemas se refiere a todos los aspectos del desarrollo de sistemas informáticos, incluyendo hardware, software e ingeniería de procesos. La ingeniería de software es parte de este proceso

Ciclo de vida del software

Definición de necesidades

- Los clientes e ingenieros definen el software a producir y las restricciones de su operación.

- Se solicitan y recopilan los requerimientos y necesidades a satisfacer.

Análisis

- Con base en las necesidades considerar restricciones, flujo y procesamiento de la información así como las arquitecturas y tecnologías más adecuadas para su construcción.

Diseño

- Construcción del sistema en papel, incluyendo toda la documentación y representaciones graficas necesarias para construir el software por un equipo de trabajo.

Codificación

- Implementar el diseño apoyándose de las herramientas de programación necesarias.
- Es importante que conforme la codificación va avanzando, se **documente en el la relación codificación-diseño**.

Pruebas

- Las pruebas de software, son los procesos que permiten verificar y revelar la calidad de un producto software. Son utilizadas para identificar posibles fallos de implementación, calidad, o usabilidad de un programa.

Validación

- Las pruebas de validación en la ingeniería de software son el proceso de revisión que el sistema de software producido cumple con las especificaciones y que cumple su cometido. Es normalmente una parte del proceso de pruebas de software de un proyecto, que también utiliza técnicas tales como evaluaciones, inspecciones, y tutoriales. La validación es el proceso de comprobar lo que se ha especificado es lo que el usuario realmente quería.

Mantenimiento y evolución

- **El mantenimiento del software** contempla la modificación de un producto de software después de la entrega para corregir averías, para mejorar funcionamiento u otras cualidades, o para adaptar el producto a nuevas utilidades y funciones.

  - **Mantenimiento correctivo:** La modificación reactiva de un producto de software se realizó después de entrega para corregir problemas descubiertos.

- **Mantenimiento adaptante:** La modificación de un producto de software se realizó después de entrega para mantener un producto de software usable un ambiente cambiante o que cambiaba.

- **Mantenimiento de perfectivo:** Modificación de un producto de software después de la entrega para mejorar funcionamiento o capacidad de mantenimiento.

- **Mantenimiento preventivo:** Modificación de un producto de software después de que entrega para detectar y para corregir averías latentes en el producto de software antes de que se conviertan en averías eficaces.

Documentación del software

- Aunque muchas veces es omitida por los principiantes y los que se dedican a producir resultados rápidos debido a que no es tan atractiva como la codificación, la documentación --al igual que el diseño-- es una marca del orgullo profesional que el programador pone en sus creaciones.

- Existen dos clases de documentación:
  - *Documentación externa*
  - *Documentación interna*

Documentación Externa

- En esta categoría están no sólo los más visibles para los usuarios tales como los manuales de operación y de instalación del software, sino también los documentos de diseño utilizados durante el desarrollo del software: copia de los requerimientos, copia del algoritmo empleado así como de las alternativas consideradas, diagramas de flujo, copia de los documentos que se utilizan para el diseño de las entradas y salidas visuales o impresas, copia de los estándares de desarrollo, listado más actual del código fuente, documentos relacionados con las modificaciones hechas al proyecto, así como notas importantes de diseño usadas por los desarrolladores durante el diseño y la implementación, entre otras.

Documentación Interna

- A diferencia de la documentación externa, la documentación interna se encuentra dentro del código mismo y es la más detallada de las dos. Puesto que es la más cercana al código, la documentación interna es la que se suele mantener más actualizada y correcta a medida que el código se modifica.

- La documentación interna es muy importante puesto que facilita grandemente la lectura y comprensión del código, tanto para el propio programador como para todos los que necesiten leerlo, y es especialmente útil en las fases de prueba y mantenimiento de los programas.

- Es importante la relación que se mantenga entre la documentación interna con el diseño y parte de la documentación externa.

- El principal contribuyente de la documentación a nivel de código no son los comentarios, sino el buen estilo de programación, el cual incluye una buena estructura de programa, el uso de un acercamiento directo y fácilmente comprensible, buenos nombres de variables y de rutinas, uso de constantes nombradas en lugar de valores literales, un esquema claro y la minimización de la complejidad del control de flujo y de las estructuras de datos.

- El código de los programas escritos con un estilo pobre de programación casi siempre es críptico.

- Aún cuando se le trate de explicar usando comentarios, su calidad no se ve mejorada: permanece oscuro, difícil de leer y de modificar.

- La única manera de clarificarlo es rescribiéndolo usando un mejor estilo de programación.

- De este modo el código será más legible, aún si no posee comentarios que lo expliquen.

- El código que es legible por sí solo se denomina código autodocumentado, y es una característica deseable para cualquier programa de software.

- Tal código descansa en el buen estilo de programación utilizado en su creación para sobrellevar la mayor parte de la documentación interna.

- En un código bien escrito, los comentarios son piezas de información realmente necesarias que complementan la legibilidad y no una carga extra que, aunque de utilidad, hace que se incremente el tamaño del código fuente.

Código autodocumentado

- Si al escribir el código se cumple con los siguientes cuestionamientos, es posible indicar que un código en C es autoexplicativo.

Funciones

- ¿El nombre describe exactamente lo que hace?
- ¿Desempeña una procedimiento o función bien definido?
- ¿Las variables y datos estructurados de cada función corresponden a dicha función?
- ¿Es obvio y claro el prototipo de cada función?

Variables

- ¿Los nombres son lo suficientemente descriptivos?
- ¿Son las variables usadas únicamente para el propósito para el cual fueron nombradas?
- ¿Se utilizan constantes nombradas en lugar de constantes literales?
- ¿La convención de nombres de identificadores empleada distingue entre los nombres de tipos de datos, los tipos enumerados, las constantes nombradas, las variables locales y las variables globales?

Flujo del programa

- ¿Es claro el flujo de ejecución a través del código?
- ¿Están las sentencias de código relacionadas agrupadas o cercanas entre sí?
- ¿Son las estructuras de control lo suficientemente simples, tal que minimizan la complejidad?
- ¿Cada ciclo desempeña una y solo una función, como debería hacerlo una rutina bien diseñada?
- ¿Se ha minimizado el anidamiento (de ciclos y rutinas por ejemplo)?

Comentarios efectivos

- Los comentarios erróneos al código pueden confundir a sus lectores, incluso si el código se ha escrito usando un buen estilo de programación. El código con este tipo de comentario es peor que el código sin comentarios. Por otro lado, si los comentarios son correctos pero sólo repiten verbosamente las sentencias de código, no añaden valor al código mismo.

- Los comentarios pobres o crípticos no son de mucha ayuda y tienden a ser mal entendidos o pasados por alto debido a que obstruyen la correcta comprensión del código.

- El comentar efectivamente no consume mucho tiempo. Los comentarios excesivos son tan malos como la existencia de pocos de ellos, por lo que lograr el equilibrio es importante.

- Puede tomar mucho tiempo escribir comentarios por dos razones comunes. La primera, el estilo de comentarios puede consumir demasiado tiempo o ser tedioso.

- Un estilo que requiera mucho trabajo mantener es un dolor de cabeza: Si los comentarios son difíciles de cambiar, no se cambiarán: se volverán imprecisos y llevarán a confusión, lo cual es peor que no tener comentarios del todo.

- El comentar podría ser difícil debido a que las palabras para describir lo que el programa está haciendo no surgen fácilmente.

- Eso usualmente es una señal de que no se entiende bien lo que hace el programa.

- El tiempo que se invierte en “comentar” es en realidad tiempo invertido en comprender mejor el programa, lo cual es tiempo que necesita invertirse sin importar si lo comentas o no.

- El ir comentando el código a medida que se escribe tiene la ventaja de que se va introduciendo mientras más frescos se tienen los detalles de programación. Comentar al final, por el contrario, consume más tiempo puesto que se deben recordar los detalles y las sutiles suposiciones de la programación, o leer de nuevo el código para comprenderlo, antes de escribir los comentarios, haciendo que estos sean menos precisos.

- Si el código requiere mucha concentración es una clara señal de que es complejo, pero una alternativa sería utilizar el algoritmo como punto de partida para la codificación e ir convirtiendo las frases en comentarios a medida que codificamos.

- Si el diseño es difícil de codificar, hay que simplificarlo antes de preocuparse del código o los comentarios.

- Si se usa un algoritmo (diagrama de flujo, pseudocódigo, etc.) para clarificar ideas, la codificación será directa y los comentarios automáticos.

- Los comentarios incrementan el tamaño de los archivos de código fuente.

  - Aunque los compiladores modernos son capaces de obviar los comentarios al momento de traducirlos a código objeto y a programas ejecutables, en algunos ambientes como la Internet donde debe enviarse una copia del programa para ser interpretado en máquinas clientes (código ASP y applets de Java y JavaScript, por ejemplo) el tiempo invertido en enviar la información extra que representan los comentarios a través de la red puede ser prohibitivo pues penaliza el desempeño de las aplicaciones, aún si éstas se ejecutan en hardware remoto eficiente. Sin embargo, eso no quiere decir que no se deba comentar el código.

- Aún así, una solución práctica para este problema es tener dos versiones del código: una comentada y otra en producción. El truco consiste en escribir un programa que filtre los comentarios de los archivos fuente (que busque y deseche del código todo el texto que se halla protegido por las secuencias sintácticas que el lenguaje usa para los comentarios) antes de pasar al proceso de compilación o implantación en los servidores de las aplicaciones remotas.

- El utilizar el algoritmo o el seudocódigo como punto de partida para la codificación tendrá como efecto colateral comentarios muy precisos.

Técnicas para comentar código

- **Comentarios por bloques:** comentarios de una o dos líneas que describan bloques de código.
- **Escribe comentarios que describan la intención del código:** en lugar de repetir el código, escribir comentarios que describen el propósito del código.
- **Enfoca tus esfuerzos de documentación en el código mismo: e**l código mismo debería ser la mejor documentación.
- **Enfoca el comentario del bloque en el por qué en lugar del cómo:** los comentarios que explican cómo se hace algo están a un nivel de programación, algo técnicos, en lugar de estar a nivel del problema.

- **Evita las abreviaturas:** los comentarios no deben ser ambiguos, sino fácilmente legibles sin el trabajo de adivinar abreviaturas.

- **Documenta las sorpresas:** si hay lago que no es obvio a partir del código mismo. Cada truco técnico que se use debería ser comentado.

- **Comenta cualquier truco que evite un error o una funcionalidad no documentada en un lenguaje o entorno:** si es un error, probablemente no esté documentado. Aún si está documentado en alguna parte, no daña el documentarlo de nuevo en tu código. Si es una funcionalidad indocumentada, debería estarlo en tu código

- **Comenta las unidades de datos numéricos:** cada variable o campo que vaya a contener cantidades numéricas que representen unidades de medida debería tener documentada la unidad de medida que representa. Así, si una variable representará longitud debería documentarse si representará centímetros, pulgadas, metros, kilómetros, etc. Si son coordenadas, si indicarán latitud, longitud o altitud, y si está en grados o radianes; No asumas que las unidades son obvias pues para un nuevo programador u otro que esté trabajando en otra parte del programa/sistema, no lo serán.

- **Coloca un comentario antes de cada estructura de control:** las estructura de control a menudo necesitan explicación, y el mejor lugar para documentarlas es justo el espacio antes de ellas. En el caso de una sentencia de selección, debe incluirse la razón para la decisión y un resumen del resultado. Si es un ciclo, debe indicarse su propósito y aspectos importantes de su ejecución.

- **Describe cada función con una o dos oraciones en la línea anterior a ella:** un comentario descriptivo breve permitirá conocer rápidamente el propósito de la función. Si tienes dificultades en crear una descripción corta para la rutina es una clara señal de que el diseño de tu programa no es tan bueno como debería.

- **Documenta los parámetros de entrada y salida:** la forma más práctica de documentar variables de entrada y salida de una función es colocar esos comentarios justo bajo la línea de encabezado de la rutina misma.

- **Comenta las limitaciones de la función:** si la función devuelve un resultado numérico, indica su precisión. Si la rutina funciona sólo para estructuras de datos de cierto tamaño, indícalo. Si se sabe de modificaciones que harán que la rutina deje de funcionar, documentarlas también. Documenta los efectos globales de las modificaciones que realiza la rutina

- **Si dentro de la rutina se modifica una variable o estructura de datos global**: debe describirse de forma explícita esta modificación.

Reutilización de código

- La reutilización de código se refiere al comportamiento y a las técnicas que garantizan que una parte o la totalidad de un programa informático existente se puedan emplear en la construcción de otro programa. De esta forma se aprovecha el trabajo anterior, se economiza tiempo, y se reduce la redundancia.

Tipos de reutilización

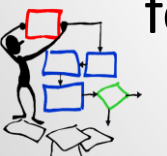
- **Reutilización oportunistas**
  - Ocurre cuando al iniciar un proyecto, el programador se da cuenta de que hay componentes existentes que se puede reutilizar.

- **Reutilización planificada**
  - Sucede cuando un equipo planea estratégicamente los diseños de componentes que serán reutilizables en futuros proyectos.

Copiar y pegar

- La manera más fácil de reutilizar código es copiarlo total o parcialmente desde el programa antiguo al programa en desarrollo. Pero es trabajoso mantener múltiples copias del mismo código, por lo que en general se elimina la redundancia dejando el código reusable en un único lugar, y llamándolo desde los diferentes programas. Este proceso se conoce como **abstracción procedimental**.

- La abstracción de procedimientos y funciones puede verse claramente en las **bibliotecas de software**, en las que se agrupan varias operaciones comunes a cierto dominio para facilitar el desarrollo de programas nuevos. Hay bibliotecas para convertir información entre diferentes formatos conocidos, acceder a dispositivos de almacenamiento externos, proporcionar una interfaz con otros programas, manipular información de manera conocida (como números, fechas, o cadenas de texto).



Reutilizar código en C

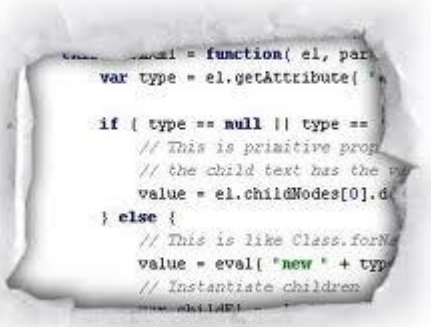
- Para que el código existente se pueda reutilizar, debe definirse alguna forma de comunicación o interfaz. Esto se puede dar por llamadas a una función o procedimiento.

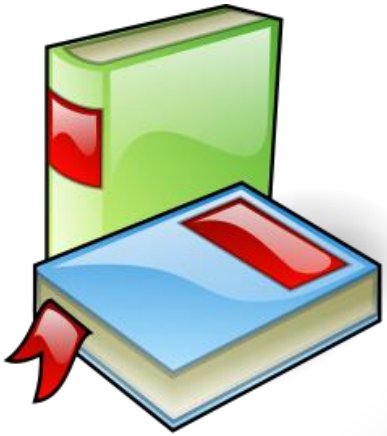
- En lenguaje C como sabemos únicamente podemos modelar los procedimientos con funciones que retornan un dato tipo *void,* estas funciones deberán estar siempre bien diseñadas para poder reutilizarlas en más programas. Si además las agrupamos según su comportamiento puede formarse nuestras propias librerías de funciones útiles para varios desarrollos distintos.

Concepto de Librería en Programación

- Es un conjunto de subprogramas utilizados para desarrollar software.

- Las bibliotecas contienen código y datos, que proporcionan servicios a programas independientes, es decir, pasan a formar parte de estos. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular.





- Algunos programas ejecutables pueden ser a la vez programas independientes y bibliotecas, pero la mayoría de estas no son ejecutables.

- Ejecutables y bibliotecas hacen referencias **(enlaces)** entre sí a través de un proceso conocido como *enlace*, que por lo general es realizado por un software denominado **enlazador**.

- Las bibliotecas o librerías, pueden ser clasificadas según el tipo de enlace que se realice para ser parte de un programa final en:
  - Bibliotecas estáticas
  - Bibliotecas dinámicas

Biblioteca estándar de C

- La biblioteca estándar de C es una recopilación de **archivos cabecera** y **bibliotecas con funciones**, **estandarizadas** por un comité de la Organización Internacional para la Estandarización (ISO), que implementan operaciones comunes, tales como las de entrada y salida o el manejo de cadenas. A diferencia de otros lenguajes como COBOL, Fortran, o PL/1, C no incluye palabras clave para estas tareas, por lo que prácticamente todo programa implementado en C se basa en la biblioteca estándar para funcionar

Creación de bibliotecas para C

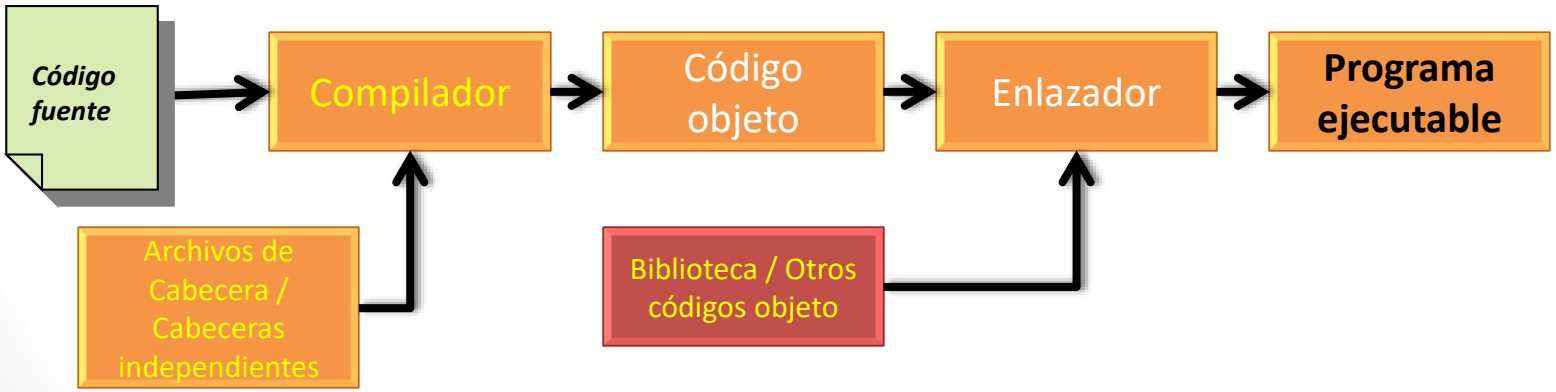
- Es posible generar biblioteca para C generando nuestros propios **archivos cabecera** y **bibliotecas con funciones**.

Archivos cabecera

- Es un archivo, que el compilador incluye al procesar algún código fuente, este contiene, normalmente, una declaración directa funciones, variables, u otros identificadores. Aquellos programadores que desean declarar identificadores estándares en más de un archivo fuente pueden colocar esos identificadores en un único header file, que se incluirá cuando el código que contiene sea requerido por otros archivos.

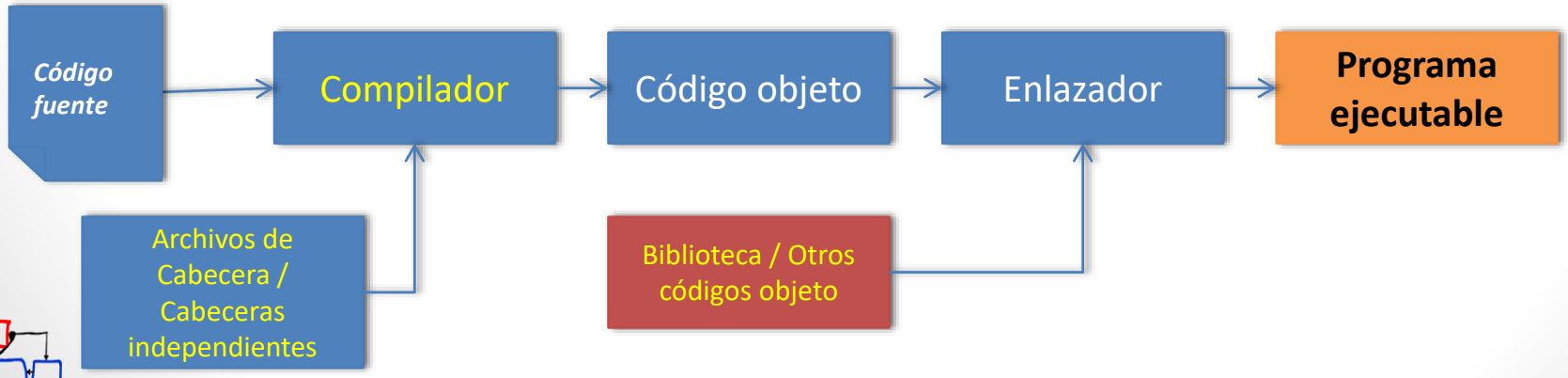
Bibliotecas con funciones

- Son los códigos fuentes que definen las funciones de los archivos de cabecera y son independientes. Estos pueden ser compilados por separados y tenerse como el código fuente original o códigos objetos capaces de ser enlazados por otros códigos fuentes que hagan uso de estas definiciones y programas.



Generación de código ejecutable

- Como se ve en la etapa de compilación de un lenguaje compilado, se obtiene un código objeto, el cuál contiene sólo la traducción del código fuente. Esto no es suficiente para ejecutar realmente el programa. Es necesario incluir los archivos de biblioteca o módulos compilados de manera independiente.



Generando una librería para C

- Generar los siguientes archivos, con los contenidos que se muestran y guardarlos con los nombres dados.

```c
#include <stdio.h>
#include "mi_libreria.h"

int main (void)
{
        int n,res;
        printf("\nIntroduce un número entero")
        scanf("%d",n);

        res=mi_funcion01(n);

        printf("\nEl resultado es: %d",res)

        return 0;
}
```

programa.c

```c
#define CONSTANTE 100

int mi_funcion01(int numero);
```

mi_libreria.h

```c
#include "mi_libreria.h"

int mi_funcion01(int numero)
{
        return numero*CONSTANTE;
}
```

def_mi_libreria.c

- Generar el código objeto de la librería

gcc def_mi_libreria.c –c

- Compilar el programa

gcc programa.c def_mi_libreria.o –o programa