Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño y programación

Índice

- Reglas de conducta
- Reglas de diseño
- Reglas de Programación
- Preguntas

Ingeniería de Software de Fuentes Abiertas/Libres

Qué reglas de conducta conocen?

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Conducta

- **Respetar.**
- **Debatir cualquier cambio.**
- **Estudiar cualquier cambio.**
- **Colaborar.**
- **Renunciar consideradamente.**
- **Documentar o Informe su trabajo.**
- **Pedir ayuda.**
- **Brindar ayuda.**
- **Sé pragmático.**
- **Saber asumir compromisos.**

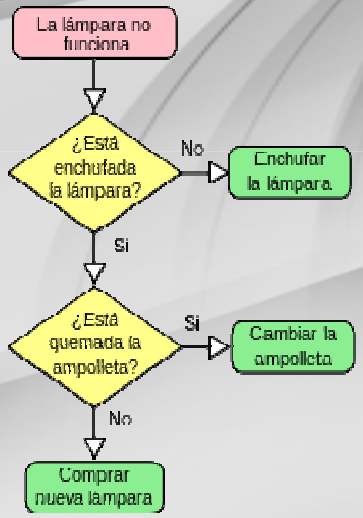
Ingeniería de Software de Fuentes Abiertas/Libres

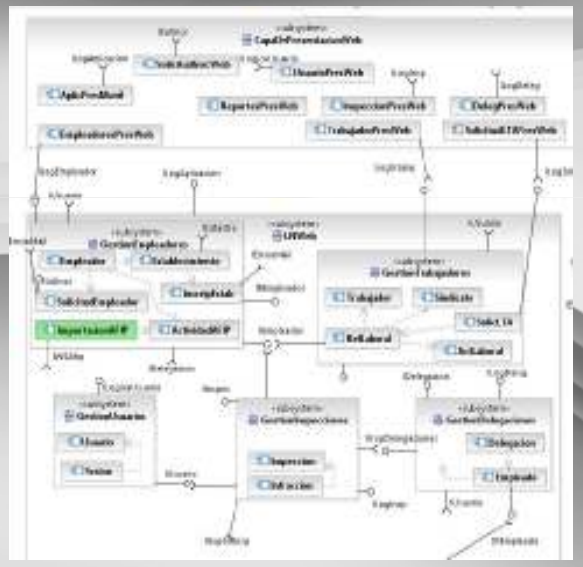
Qué reglas diseño conocen?

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Modularidad: Escribe partes simples, Conectadas por interfaces simples.





Ingeniería de Software de Fuentes Abiertas/Libres

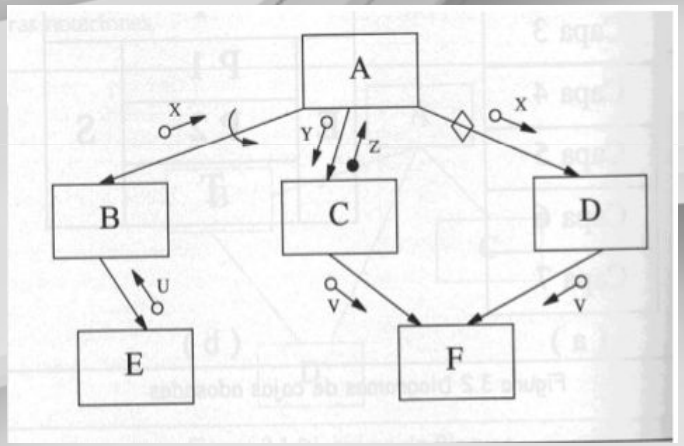
Reglas de diseño

Regla de Claridad: Ser CLARO es mejor que ser INGENIOSO

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

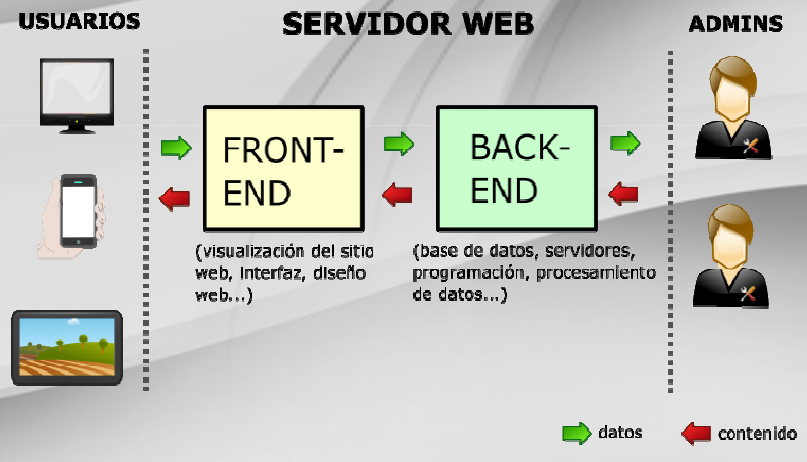
Regla de Composición: Diseñe Programas para que se conecten con otros programas



Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Separación: Separa las reglas del funcionamiento, las interfaces de los mecanismos.

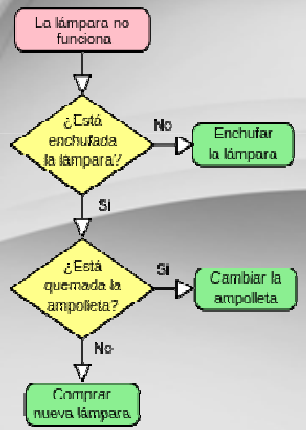


Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Simplicidad: Diseña para la simplicidad, añade complejidad solo cuando sea necesario

Regla de Parsimonia: Escribe un programa complejo solo cuando sea evidente que no hay otra forma de hacerlo

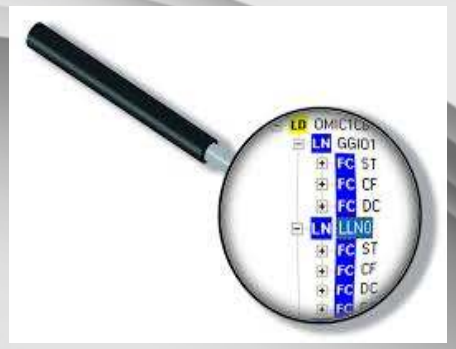


Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Transparencia: Diseña para la visibilidad, inspección y corrección de errores

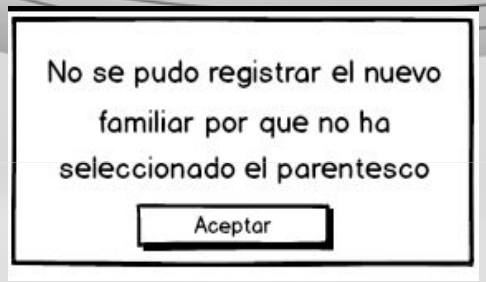
```java
package trilcejf;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletEnlace extends HttpServlet{
    public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException{

        response.setContentType("text/html");

        PrintWriter out=response.getWriter();

        out.println("<html>");
        out.println("<head><title>ServletEnlace</title></head>");
        out.println("<body>");
        out.println("<h2>Vienes de pulsar el enlace \"Púlsame\"</h2>");
        out.println("<img src=\"/Prueba/tomcat.gif\">");
        out.println("</body>");
        out.println("</html>");
    }
}
```

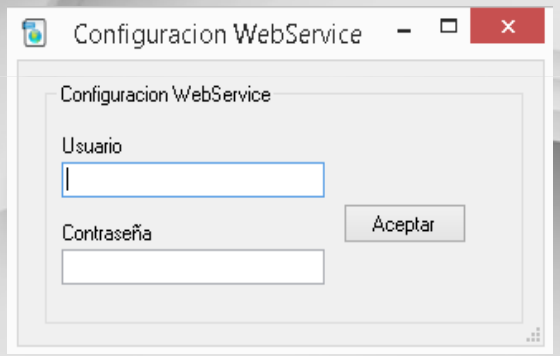


Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Robustez: La robustez es hija de la transparencia y la simplicidad.

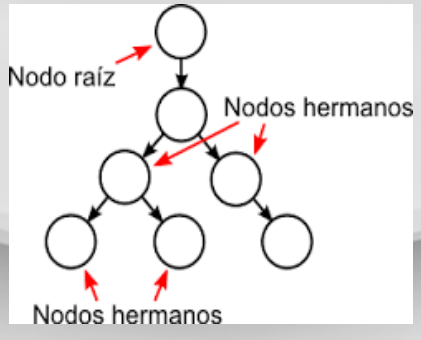


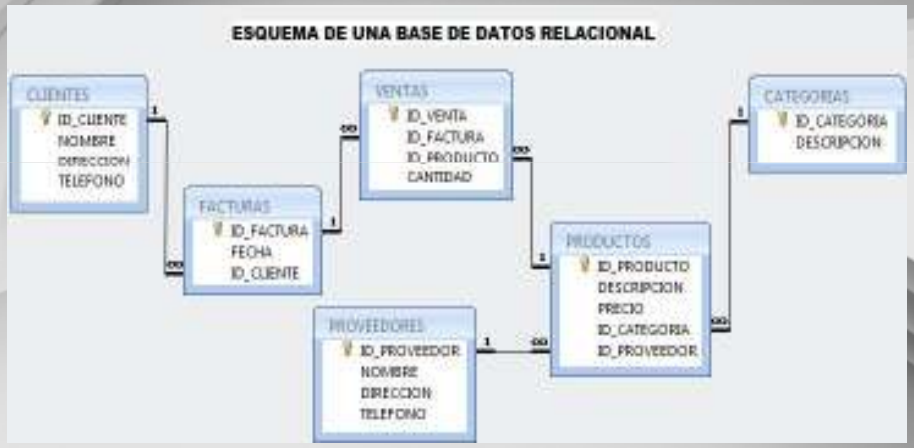


Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Representación: Convierte el conocimiento en datos, para que la lógica sea sencilla y robusta.

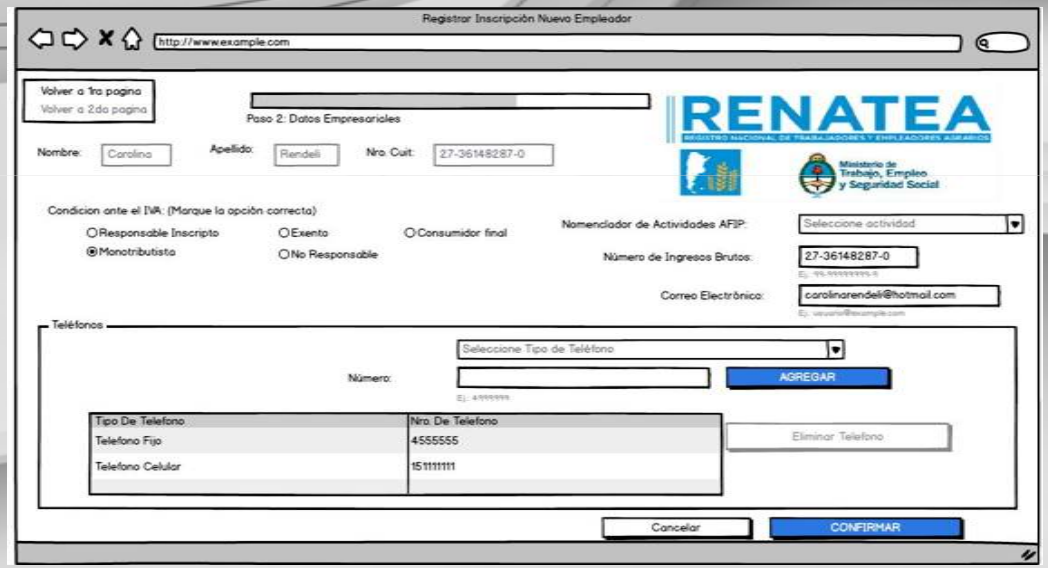




Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Mínima Sorpresa: En diseño de interfaz, haz siempre lo menos sorprendente.



Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Silencio: Cuando un programa no tenga nada que decir, no debería decir nada.

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Reparación: Cuando tengas que mostrar un error, Falla estridentemente y lo antes posible

Ingeniería de Software de Fuentes Abiertas/Libres

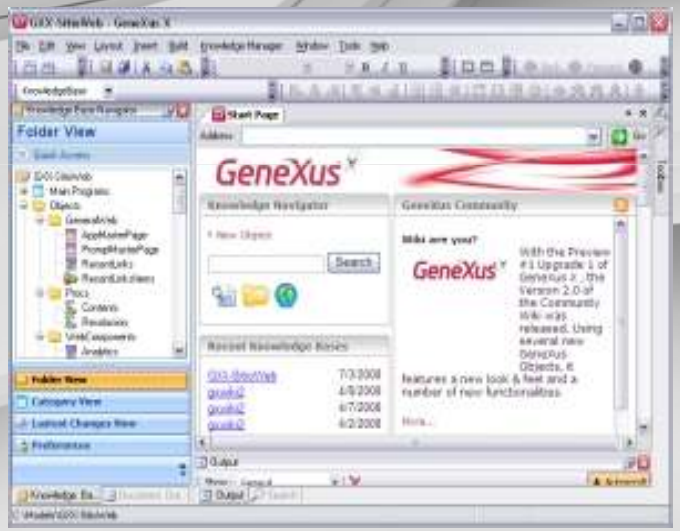
Reglas de diseño

Regla de Economía: El tiempo del programador es caro, conservelo sobre el tiempo de la máquina

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

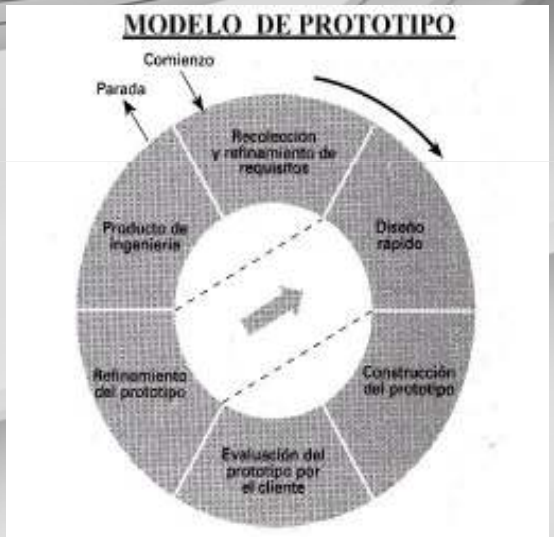
Regla de Generación: Evite hacer cosas a mano, use programas que escriban programas siempre que puedan



Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Optimización: Prototipa antes de pulir, haz que funcione antes de optimizarlo



Ingeniería de Software de Fuentes Abiertas/Libres

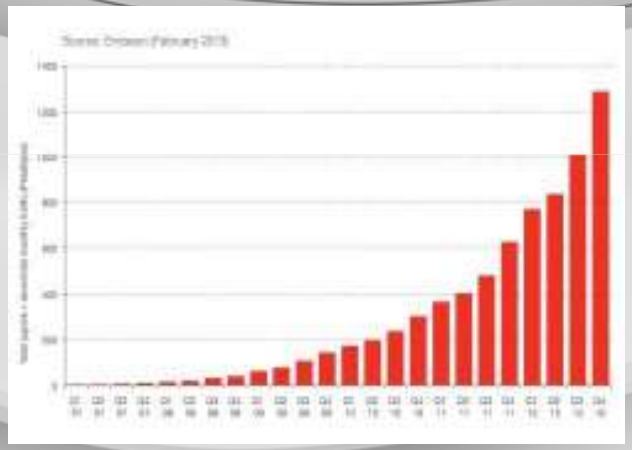
Reglas de diseño

Regla de Diversidad: Desconfíe de todas las afirmaciones (“Esta es la unica forma correcta”)

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de diseño

Regla de Extensibilidad: Diseña para el futuro, Estará aqui más rápido de lo que piensas



```
namespace clase2_operadores_estructuras_arreglos
{
    class operadores
    {
        static void Main(string[] args)
        {
        //Declaro las variables necesarias
        int a,b;
        double suma, resta, producto, div, modulo;

        //Asignando valores a las variables
        a = 45;
        b = 100;

        //Realizando las operaciones aritméticas
        suma = a + b;

        //Bloque de codigo para mostrar resultados
        Console.WriteLine("*** OPERACIONES MATEMATICAS ***\n");
        Console.WriteLine("Suma = {0}", suma);
        Console.WriteLine("Operaciones realizadass con exito");
        Console.ReadKey();
        }
    }
}
```

Ingeniería de Software de Fuentes Abiertas/Libres

Qué reglas de programación conocen?

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Programación

```
if (hours < 24 && minutes < 60 && seconds < 60)
{
    return true;
}
else
{
    return false;
}
```

```
if  ( hours   < 24
   && minutes < 60
   && seconds < 60
)
{return    true
;}         else
{return   false
;}
```

```
if(hours<24 && minutes<60 && seconds<60){return true;}else{return
false;}
```

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Programación

- **Propias de cada comunidad**
- **Ayuda a los programadores de la comunidad a comprender el código de forma más sencilla**
- **Trata de evitar la introducción de nuevos errores (típicos)**

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Programación

Ejemplo: coreutils

SI

```
while (expr)
    single_line_stmt ();
```

NO

```
while (true)
    /* comment... */
    single_line_stmt ();
```

SI

```
while (true)
  {
        /* explanation... */
    single_line_stmt ();
  }
```

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Programación

Ejemplo: coreutils

SI

```
while (expr)
    single_line_stmt ();
```

NO

```
while (true)
    /* comment... */
    single_line_stmt ();
    other_stmt ();
```

SI

```
while (true)
  {
        /* explanation... */
    single_line_stmt ();
  }
```

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Programación

Ejemplo: coreutils

EXCEPCIÓN

```
if (expr)
    error (0, 0, _("a diagnostic that would make this line"
                   " extend past the 80-column limit"));
```

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Programación

Indentación:

```
switch (suffix) {
case 'G':
case 'g':
        mem <<= 30;
        break;
case 'M':
case 'm':
        mem <<= 20;
        break;
case 'K':
case 'k':
        mem <<= 10;
        /* fall through */
default:
        break;
}
```

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Programación

Longitud de línea: (80 columnas)

```
/* Can display any amount of data, unlike the Unix version, which
uses
   a fixed size buffer and therefore can only deliver a limited
number
   of lines.

   Original version by Paul Rubin <phr@ocf.berkeley.edu>.
   Extensions by David MacKenzie <djm@gnu.ai.mit.edu>.
   tail -f for multiple files by Ian Lance Taylor <ian@airs.com>.
   inotify back-end by Giuseppe Scrivano <gscrivano@gnu.org>.  */

#include <config.h>

#include <stdio.h>
#include <assert.h>
#include <getopt.h>
#include <sys/types.h>
#include <signal.h>
```

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Programación

Espacios y llaves:

```
if (x is true) {
        we do y
}
```

```
int function(int
x)
{
    // body
}
```

```
switch (action) {
case KOBJ_ADD:
        return "add";
case KOBJ_REMOVE:
        return "remove";
case KOBJ_CHANGE:
        return "change";
default:
        return NULL;
}
```

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Programación

Nombres de variables y funciones:

```
int ThisVariableIsATemporaryCounter = 0;
```

```
int count_active_users()
```

```
int tmp = 0;
```

```
int cau()
```

Ingeniería de Software de Fuentes Abiertas/Libres

Reglas de Programación

Funciones:

```
int do_one_thing() {
    // the one thing
}
```

```
int do_EVERYTHING() {
    // 1000+ lines of code
}
```

Proyecto Unix

- La filosofía de Unix no es un método de diseño formal.
- La filosofía de Unix es de abajo (experiencia) hacia arriba (teoría). Es pragmático y basado en la experiencia.
- La filosofía de Unix se encuentra en el conocimiento implícito mitad-reflexivo, la experiencia que la cultura Unix transmite.

Proyecto Unix

Rob Pike, uno de los grandes maestro de C, propuso las siguientes reglas de la programación:

- **Regla 1**: ***Cuellos de botella.*** No se puede decir dónde un programa está gastando más tiempo, producto de un cuello de botella (Bottlenecks) . Los Bottlenecks ocurren en lugares inesperados, así que no trate de adivinar o de suponer estas trabas de velocidad hasta que haya demostrado con total seguridad dónde está el bottlenecks.
- ***Regla 2: Medir***. No suponga que una parte del código es lenta hasta que se mida.
- **Regla 3:** ***Algoritmos sofisticados***. Los algoritmos sofisticados son lentos cuando "n" es pequeño y "n" es generalmente pequeño. Los algoritmos sofisticados tienen constantes grandes. Hasta que sabes que "n" con frecuencia va a ser grande, no se divierte. (Incluso si "n" se hace grande, use primero la Regla 2.)
- **Regla 4**: ***Simplicidad***. Utilice algoritmos simples, así como estructuras de datos simples.
- **Regla 5.** ***Los datos dominan***. Si ha elegido las estructuras de datos adecuadas y tienen las cosas bien organizadas, los algoritmos casi siempre serán evidentes, entendibles y simples. Las estructuras de datos, no los algoritmos, son fundamentales para la programación.

Regla 6. No existe Regla 6. // Ken Thompson: En caso de duda, use fuerza bruta.

Proyecto GNU

El Proyecto GNU considera las normas publicadas por otras organizaciones como sugerencias, no como órdenes

- Seguir los estándares publicados es conveniente para los usuarios; significa que sus programas o secuencias de comandos funcionan de forma más portátil.
- Pero no seguimos rigurosamente ninguna de estas especificaciones, y hay puntos específicos sobre los cuales decidimos no seguirlas, para que el sistema GNU sea mejor para los usuarios.
- No rechace una característica nueva, ni elimine una antigua, simplemente porque un estándar dice que está "prohibido" o "desaprobado".

GNU Coding Standars

- Formato del Código Fuente:
  - **Longitud de línea de código**: La longitud de las líneas de código >= 79 caracteres.
  - **Definición de Funciones:**
    - La apertura de llaves (“{“) de la definición de una función debe ser en la columna 1 de la línea.
    - Evite poner llave abierta (“{“), paréntesis abierto (“(“) o corchete abierto (“[“) en la columna 1 cuando no inicien una definición de función.
    - También es importante que los nombres de las definiciones de funciones inicien en la columna uno.
  - **Consistencia de estilos**: Mezcla de estilos dentro de un programa tiende a parecer feo.
  - **Utilización de espacios:** Nos resulta más fácil leer un programa cuando tiene espacios antes de los paréntesis abiertos y después de las comas.
  - **Operadores diferentes**: Trate de evitar tener dos operadores de precedencia diferente en la misma línea.
  - **Uso de paréntesis**: Use paréntesis para separar anidamiento y que sea más fácil de entender.

GNU Coding Standars

- Comentar código
  - **Propósito del archivo**: Nombre del archivo y una línea o dos sobre el propósito general del archivo.
  - **Idioma de comentarios**: Escriba los comentarios de un programa GNU en inglés, porque el inglés es el único idioma que casi todos los programadores de todos los países pueden leer.
  - **Funciones**: Ponga un comentario sobre cada función diciendo qué hace la función, qué tipo de argumentos obtiene, qué significan los posibles valores de los argumentos y para que se utiliza. También el significado de lo que la función devuelve, en caso de retornar algún valor.
    - Nombres de los argumentos para hablar de los valores de los argumentos.
  - **Formato de comentarios**: Por favor ponga dos espacios después del final de una oración en sus comentarios, para que los comandos de sentencia de Emacs funcionen.

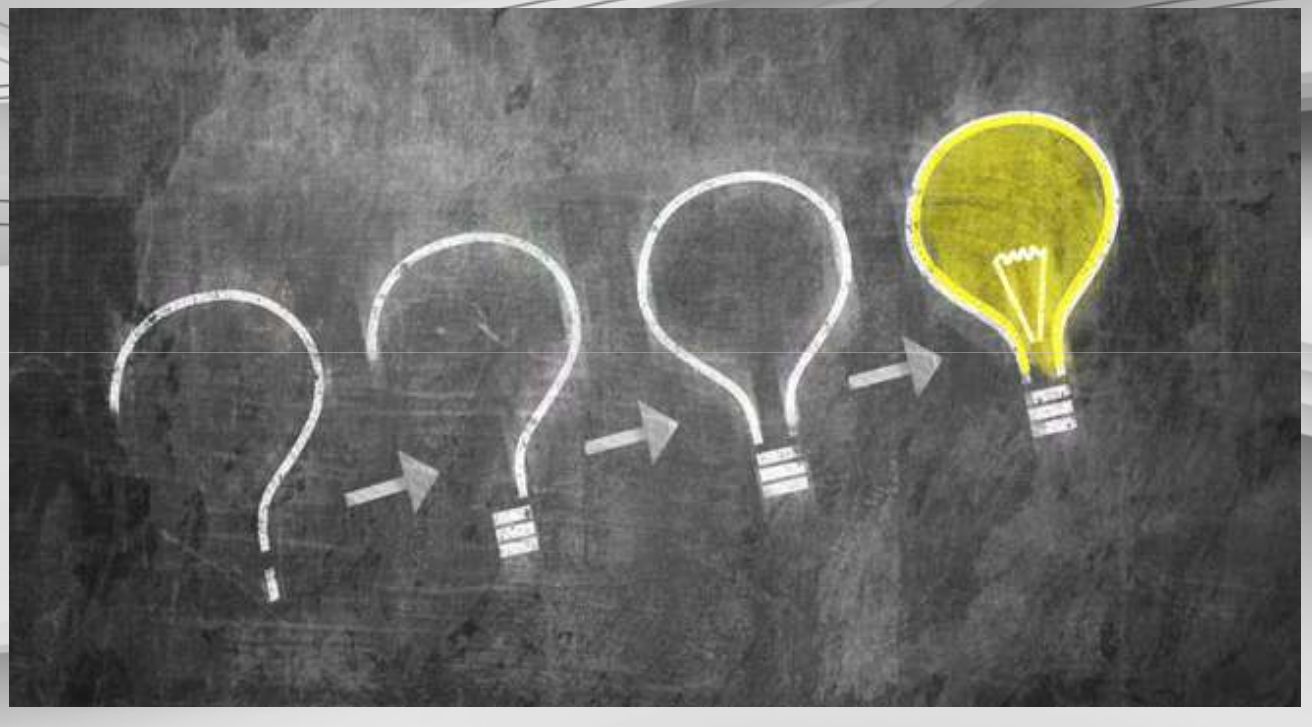
GNU Coding Standars

- Uso limpio de los constructores C
  - **Declaración de tipos de objeto**. Por favor declare explícitamente los tipos de todos los objetos.
  - **Variable local separada para cada propósito distinto**. Dejor declarar una variable local separada para cada propósito distinto, y darle un nombre que sea significativo.
  - **Declaración de múltiples variables**. No declare múltiples variables en una declaración que se extiende por líneas. Inicie una nueva declaración en cada línea.
  - **Frenos de if-else**. Cuando tiene una sentencia if-else anidada en otra sentencia if, siempre coloque frenos alrededor del if-else
  - **Sentencia else-if.** Si tiene una sentencia if anidada dentro de una sentencia else, escriba else if en una línea
  - **Separación estructura y declaración**. Declare la etiqueta de estructura por separado y luego utilícela para declarar las variables o typedefs.
  - **Asignación dentro si-condiciones**. Trate de evitar las asignaciones dentro de si-condiciones (las asignaciones dentro de-las condiciones están bien).

GNU Coding Standars

- Nombrar variables, funciones y archivos
  - **Nombres significativos**. Busque nombres que proporcionen información útil sobre el significado de la variable o función.
  - **Nombres cortos**. Los nombres de variables locales pueden ser más cortos, ya que se utilizan sólo en un contexto.
  - **Abreviaciones**. Trate de limitar el uso de abreviaturas en los nombres de símbolos.
  - **Formato de nombre**. Utilice subrayados para separar las palabras de un nombre, para que los comandos de palabra de Emacs puedan ser útiles dentro de ellos. Stick para minúsculas; Reserva mayúscula para macros y constantes de enumeración, y para prefijos de nombres que siguen una convención uniforme.
  - **Banderas**. Las variables que indican si se han especificado las opciones de la línea de comandos deben ser nombradas después del significado de la opción, no después de la letra de opción. Un comentario debe indicar el significado exacto de la opción y su letra.

Preguntas



Agradecimientos

MUCHAS GRACIAS!!