SISTEMA DE INVENTARIO Y FACTURACION (SIF)

Especificaciones de Requerimientos del Sistema

Documento Estándares de Codificación

Equipo:

Alberto Gallardo

Wiliam Morales

Isaack Nogales

**La apariencia Código**

Los estilos de programación comúnmente hacer frente a la apariencia visual del código fuente, con el objetivo de exigir menos esfuerzo cognitivo humano para extraer información sobre el programa. Software ha sido durante mucho tiempo disponible que el código fuente automáticamente formatos, codificadores dejando de concentrarse sobre los nombres, la lógica, y más técnicas. Con carácter práctico, utilizando un equipo a la fuente de código de formato ahorra tiempo, y es posible ejecutar a continuación, todo en estándares empresa SYMFONY.

**Sangría**

Estilos de sangría ayudar a identificar el flujo de control y bloques de código. En algunos lenguajes de programación indentación se utiliza para delimitar bloques lógicos de código, la sangría correcto en estos casos es más que una cuestión de estilo. En otros idiomas sangría y los espacios en blanco no afecta la función, aunque lógica y la sangría constante hace que el código sea más legible.

if (horas <24 & & minutos <60 & & segundos <60)

(

**return** true;

)

más

(

**return** false;

)

o

if (horas <24 & & minutos <60 & & segundos <60) (

**return** true;

Else ()

**return** false;

)

if (horas <

24 & & minutos <

60 & & segundos <

60)

**(Return** true

;) Else

**(Return** false

;)

Los dos primeros ejemplos son probablemente mucho más fácil de leer porque son sangría en una forma establecida (un "colgar el apartado" estilo). Este estilo de sangría es especialmente útil cuando se trabaja con múltiples construcciones anidadas.

Symfony utiliza sangría para indicar estructuras de control, para corregir el sangrado es *obligatorio.* De esta manera, la necesidad de poner entre paréntesis con llaves ((y)) se elimina. Por otra parte copiar y pegar el código Symfony se puede llevar a problemas, ya que el nivel de sangrado del código de pegado no puede ser el mismo que el nivel de sangría de la línea actual. reformatear Tal es tedioso hacerlo a mano, pero algunos editores de texto y entornos de desarrollo para hacerlo automáticamente. También hay problemas cuando el código Symfony podría quedar inservible cuando se publiquen en un foro o una página web que elimina los espacios en blanco, aunque este problema se puede evitar si es posible incluir código en los espacios en blanco de conservación de etiquetas como "<pre> ... </ pre> "(para [HTML](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/HTML&rurl=translate.google.co.ve&twu=1&usg=ALkJrhjhelwmaAAKwomA-1lys_kS0-gy1g) ), "[code]" ... "[/ Code]" (para [bbcode](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Bbcode&rurl=translate.google.co.ve&twu=1&usg=ALkJrhit1V9tiDgo-dX2xzcIHswZBBZ8QA) ), etc

**La alineación vertical**

A menudo es útil para alinear verticalmente los elementos similares, para que los errores generados por error-más evidente. Compare:

búsqueda $ = array ('a', 'b', 'c', 'd', 'correo');

reemplazo de $ = array ('foo', 'bar', 'baz', 'quux');

*/ / Otro ejemplo:*

$ valor = 0;

anothervalue $ = 1;

yetanothervalue $ = 2;

con:

búsqueda $ = array ('a', 'b', 'c', 'd', 'correo');

reemplazo de $ = array ('foo', 'bar', 'baz', 'quux');

*/ / Otro ejemplo:*

$ valor = 0;

anothervalue $ = 1;

yetanothervalue $ = 2;

El último ejemplo hace dos cosas intuitivamente claro que no estaban claras en la antigua:

* los términos de búsqueda y reemplazo están relacionados y coinciden: no son variables discretas;
* existe un término más búsqueda que existen términos de sustitución. Si esto es un error, es ahora más posibilidades de ser descubierto.

Argumentos en contra de la alineación vertical generalmente afirman dificultades para mantener la alineación. dificultades se pueden eliminar utilizando un editor de código fuente que soporta los tabs

**Espacios**

En aquellas situaciones donde algunos espacios en blanco requiere la mayoría de las gramáticas de las lenguas del formato libre, no les preocupa la cantidad que aparece. Estilo relacionados con los espacios en blanco se utiliza comúnmente para mejorar la legibilidad .

Por ejemplo, comparar los siguientes ejemplos sintácticamente equivalente de código C.

int i;

for (i = 0; i <10; i + +) (

printf ("% d", i \* i + i);

)

versus

int i;

for (i = 0; i <10; i + +) (

printf ("% d", i \* i + i);

)

o incluso

int i;

for (i = 0; i <10; i + +) (

printf ("% d", i \* i + i);

)

Otra solución, mucho más flexible para la distribución de código, es el uso de las tabulaciones de sangría lógico (permitiendo que cada codificador para ajustar a su tamaño preferido) y espacios para la alineación.

Class MiClase (/ / comentario en consonancia con otras observaciones con el espacio

foobar (int Qux, / / decl foobar lógicamente sangría con la lengüeta

quux int); / / quux int); se envía a la muesca una misma lógica con la lengüeta

/ / Y alineado con el espacio, como la presente línea de comentario

);

**Nombre, la lógica, y más técnicas**

**Los nombres de variables más adecuadas**

opciones adecuadas para nombres de variables se consideran la piedra angular de un buen estilo.Mal llamada variables que el código sea más difícil de leer y entender.

Por ejemplo, considere lo siguiente pseudocódigo fragmento:

Debido a la elección de los nombres de las variables, la función del código es difícil de resolver. Sin embargo, si los nombres de variable se hacen más descriptiva:

**get** abc  **if** a < 24 **and** b < 60 **and** c < 60 **return** true **else return** false

**get** hours minutes seconds  **if** hours < 24 **and** minutes < 60 **and** seconds < 60 **return** true **else return** false

**get** hours minutes seconds  **if** hours < 24 **and** minutes < 60 **and** seconds < 60 **return** true **else return** false

**Los valores booleanos en las estructuras de decisión**

Algunos programadores indican que las estructuras como los anteriores, cuando el resultado de la decisión no es más que el cálculo de un [booleano](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Boolean_datatype&rurl=translate.google.co.ve&twu=1&usg=ALkJrhhxBy_jRNTo2CCe2X1CX-ThyiGIIQ) de valor, son excesivamente prolijo e incluso propenso a errores. Prefieren que la decisión en el propio cómputo, así:

return (hours < 24) && (minutes < 60) && (seconds < 60);

La diferencia es totalmente de estilo, debido a la optimización de [los compiladores](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Compiler&rurl=translate.google.co.ve&twu=1&usg=ALkJrhgPMzu8gOvgazH7FnnTPLTtXW_XRA) pueden producir idéntico [código objeto](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Object_code&rurl=translate.google.co.ve&twu=1&usg=ALkJrhghVJew_cTHdu5ZCyhsfMnismNwUw) para ambas formas. Sin embargo, estilísticamente, los programadores no están de acuerdo que la forma es más fácil de leer y mantener.

Los argumentos a favor de la forma más larga incluyen: entonces es posible establecer una línea de punto de interrupción por sobre una rama de la decisión; otras líneas de código se podría agregar a una rama sin [refactorización](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Refactoring&rurl=translate.google.co.ve&twu=1&usg=ALkJrhjkvZcIVbWDbJk5X6T1AqfjVdsLiA) la línea de retorno, lo que aumentaría las posibilidades de errores se están introduciendo, la forma más corta siempre permitiría un depurador al paso a una línea "después de la prueba", donde las variables que intervienen son todavía en su alcance.

**Mano izquierda comparaciones**

En las lenguas que utilizan un símbolo (por lo general un único signo de igual ( = )) para [la asignación](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Assignment_(computer_science)&rurl=translate.google.co.ve&twu=1&usg=ALkJrhi-j4xSwE9BvIqHLOK1tXL3FrflAg)y el otro (normalmente dos signos de igual ( == ) para [la comparación](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Relational_operator&rurl=translate.google.co.ve&twu=1&usg=ALkJrhh2ZXEqd06gkhBilRlqSNinTbFfyA) (por ejemplo [C](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/C_(programming_language)&rurl=translate.google.co.ve&twu=1&usg=ALkJrhgM3_Wu43En5yEoMKksTxHX8FCfqA) / [C + +](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/C%252B%252B&rurl=translate.google.co.ve&twu=1&usg=ALkJrhjfe7b_VUF51Et1V6Il0M0Dr14rzA) , [Java](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Java_(programming_language)&rurl=translate.google.co.ve&twu=1&usg=ALkJrhieITwPk5_8xgTjz57x9V_xOhs1fg) ,[PHP](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/PHP&rurl=translate.google.co.ve&twu=1&usg=ALkJrhjdYLt-v_YVH6cMGLaBqOOCBbg72Q) , [Perl](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Perl&rurl=translate.google.co.ve&twu=1&usg=ALkJrhjpYK-QjtStM6cDMjP9dJR-JgaX9A) contexto numérico, y la mayoría de idiomas en los últimos 15 años), y donde las tareas pueden hacerse dentro de las estructuras de control, existe una ventaja a la adopción de la mano izquierda al estilo de comparación: poner a constantes o expresiones a la izquierda en cualquier comparación. [[1]](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Coding_standards&rurl=translate.google.co.ve&twu=1&usg=ALkJrhihG8vQq5F2o0or7b0jMqA9Igt2AA#cite_note-phpcook-0) [[1]](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://c-faq.com/style/revtest.html&rurl=translate.google.co.ve&twu=1&usg=ALkJrhj4Q4DX4Z-zVMCUnsWVrtI6FLtDMg)

Éstos son tanto de izquierda como de ocasión comparación estilos correctos, aplicados a una línea de [Perl](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Perl&rurl=translate.google.co.ve&twu=1&usg=ALkJrhjpYK-QjtStM6cDMjP9dJR-JgaX9A) código. En ambos casos, se compara el valor en la variable $a 42 en contra, y si coincide, ejecuta el código en el bloque posterior.

if ($ a == 42) (...) *# A mano derecha comparación comprobar si $ a es igual a 42.*

if ($ a == 42) (...) *# Refundición, usando la mano izquierda al estilo de comparación.*

La diferencia se produce cuando un desarrollador accidentalmente tipos = en lugar de == :

if ($ a = 42) (...) *# asignación accidental que a menudo es difícil depurar*

si (42 = $ a) (...) *# tiempo de compilación de error indica la fuente del problema*

El (la derecha) la primera línea contiene ahora un sutil defecto potencialmente: más que el comportamiento anterior, ahora se establece el valor de $a ser 42, y luego siempre se ejecuta el código en el bloque siguiente. Como este es sintácticamente legítimas, el error puede pasar inadvertida por el programador, y su software se venderá con un error.

La segunda línea (la izquierda) contiene un error de semántica, como valores numéricos no pueden ser asignados. Esto resultará en un mensaje de diagnóstico que se genera cuando el código se compila, por lo que el error no puede pasar desapercibido por el programador.

Algunos idiomas tienen incorporado protecciones contra las cesiones involuntarias. Java y [C #](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/C_Sharp_(programming_language)&rurl=translate.google.co.ve&twu=1&usg=ALkJrhie83BMTs1ucgr7aeEcTvjVCvGbvw) , por ejemplo, no admiten la conversión automática a booleano para [precisamente por esta razón](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/C_Sharp_(programming_language)&rurl=translate.google.co.ve&twu=1&usg=ALkJrhie83BMTs1ucgr7aeEcTvjVCvGbvw#Features) .

El riesgo también puede ser mitigado por el uso de [herramientas de análisis estático](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis&rurl=translate.google.co.ve&twu=1&usg=ALkJrhh5SJM40nDbssD6dYJDiJnetzpSCw) que puede detectar este problema (por ejemplo, [pelusa](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Lint_(software)&rurl=translate.google.co.ve&twu=1&usg=ALkJrhh5RqLbUuXpZAFTef_0MUzbycO5Mw) ).

**Bucles y estructuras de control**

El uso de estructuras de control lógico para recorrer se suma a un buen estilo de programación. Ayuda a alguien que lee el código para entender la secuencia del programa de ejecución (en[programación imperativa](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Imperative_programming&rurl=translate.google.co.ve&twu=1&usg=ALkJrhgihEhF1NgpRf0uAFetB_TnRklJFA) idiomas). Por ejemplo, en pseudocódigo:

i = 0  **while** i < 5 **print** i \* 2 i = i + 1 **end while**  **print** "Ended loop"

i = 0  **while** i < 5 **print** i \* 2 i = i + 1 **end while**  **print** "Ended loop"

El fragmento anterior obedece a las directrices de nomenclatura y sangría estilo, pero el siguiente uso de la "para" construir el código hace mucho más fácil de leer:

**for** i = 0, i < 5, i=i+1 **print** i \* 2  **print** "Ended loop"

**for** i = 0, i < 5, i=i+1 **print** i \* 2  **print** "Ended loop"

En muchos idiomas, el uso frecuente "para cada elemento en un rango de" patrón puede reducirse a:

**for** i = 0 **to** 5 **print** i \* 2  **print** "Ended loop"

**for** i = 0 **to** 5 **print** i \* 2  **print** "Ended loop"

En [rizado lenguajes de programación soporte](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Curly_bracket_programming_languages&rurl=translate.google.co.ve&twu=1&usg=ALkJrhhk_mi1oaThUJCkAklsSeoiKLPziQ) , se ha convertido en común de la documentación de estilo para exigir que, incluso cuando opcional, llaves colocarse después de todo [control de flujo](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/Control_flow&rurl=translate.google.co.ve&twu=1&usg=ALkJrhgqqPaxAQidTqJA9ypgPHJWpeTMwA) de construcciones.

**for** (i = 0 **to** 5) { **print** i \* 2; }  **print** "Ended loop";

**for** (i = 0 **to** 5) { **print** i \* 2; }  **print** "Ended loop";

Esto evita errores flujo del programa-que puede llevar mucho tiempo de localizar, por ejemplo, cuando un punto y coma final se introduce al final de la construcción (un error común):

for (i = 0; i <5; + + i);

printf ("% d **\ n",** i \* 2); */ \* Los cueros sangría incorrecto el hecho de que esta línea no es parte del cuerpo del bucle.* *\* /*

printf ("circuito cerrado");

... O en otra línea se añade antes de la primera:

for (i = 0; i <5; i + +)

fprintf ('log', "bucle llegado% d **\ n",** i);

printf ("% d **\ n",** i \* 2); */ \* Los cueros sangría incorrecto el hecho de que esta línea no es parte del cuerpo del bucle.* *\* /*

printf ("circuito cerrado");

**Lista**

En caso de elementos de una lista se colocan en líneas separadas, a veces se considera una buena práctica para agregar el elemento separador después del último punto, así como entre cada punto, al menos en aquellas lenguas en hacerlo con el apoyo de la sintaxis (por ejemplo, , [C](http://translate.googleusercontent.com/translate_c?hl=es&langpair=en%7Ces&u=http://en.wikipedia.org/wiki/C_(language)&rurl=translate.google.co.ve&twu=1&usg=ALkJrhjQndhpskMi_Y0kNmtybI29F08Tqw) )

const char \* array [] = (

"Item1",

"Item2",

"Item3", / \* todavía tiene la coma después de \* /

);