

Coloquios Abiertos

Programando con estilo

¿Escribo un programa o un jeroglífico?



Jueves 15 de Abril de 2004
Edif. Politécnico
Salón de grados, 18:30h

J. Baltasar García Perez-Schofield

<http://trevinca.ei.uvigo.es/~ca/>

<http://trevinca.ei.uvigo.es/~jgarcia/ca/>

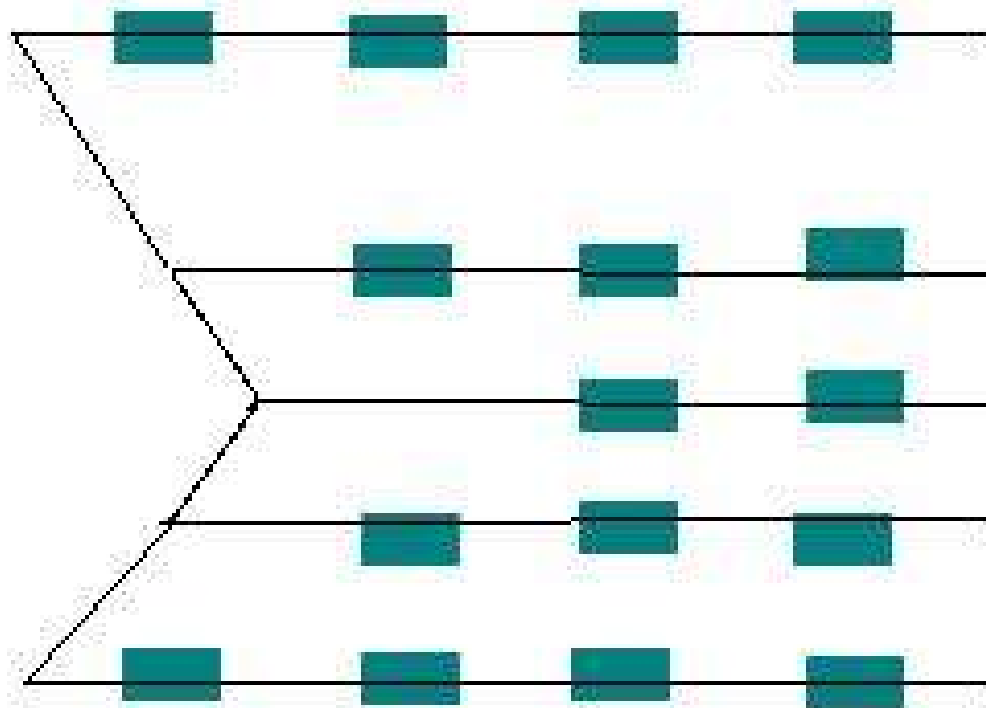
Introducción (I): La crisis del Software

- Se emplea más tiempo en mantener un programa que en crearlo por primera vez.
- El mantenimiento implica corrección de errores y ampliación de funcionalidad.
- El mantenimiento implica, por tanto, un profundo estudio del código fuente.
- El software que no se mantiene es aquel que no se usa.

Introducción (II): Manteniendo el software

- Es realmente difícil que un solo programador mantenga una aplicación determinada a lo largo de toda su vida.
- Otros programadores deben ser capaces de leer el código
- Otros programadores deben poder aprender leyendo código.

Agrupación básica recursiva de cualquier programa



Reglas generales de estilo en el código fuente

Normas generales de estilo en el código

- Nombres de identificadores
- Comentarios
- Disposición de la secuencialidad del programa
- Expresiones
- Disposición de los elementos de control (apertura-cierre de funciones, apertura-cierre de bloques)
- Disposición de los controles de flujo del programa (if, while).

Nombres de identificadores

- Tan cortos como sea posible, pero ... tan informativos como sea posible.
- La nueva técnica de moda (Java):
 - nombreDeIdentificadorLargo
- Otras técnicas (C++):
 - nombre_de_identificador_largo

Nombres de identificadores

- Por ejemplo:
- x, n, i -> bucles o argumentos de funciones simples.
- numCaracteres
 - numeroCaracteres
 - numCars
- nombreUsuario
 - nomDeUsuario
 - nomUsr

Comentarios

- Un comentario:
 - Debe introducirse sólo si es útil
 - Debe explicar, no complicar
 - No debe insultar la inteligencia del lector

```
int areaRectangulo = lado1 * lado2 // cálculo del area
x = PI * r * r; // cálculo del área del círculo
// PI es 3.1415927
```

Comentarios (II)

- Dos tipos básicos de comentarios:
 - "Encima"
 - Los más recomendables (explican un bloque)
 - "A la derecha"
 - Siempre cortos
 - Cuidado con los márgenes (no pasar de 80).

```
{ Cálculos previos al rendering }
```

```
areaRectangulo := lado1 * lado2 { en cms }
```

Secuencialidad

- Dispónganse las instrucciones de un programa en párrafos
- Cada párrafo puede llevar un comentario "arriba"
- Nunca una función debe tener más texto que una página impresa, a no ser que se trate de acciones repetitivas.
- Trátese de seguir el esquema: inicialización, proceso (normalmente, un bucle), finalización
- Cortar las líneas antes de la columna 80

Secuencialidad del programa

- Estructura básica:

```
iterator it = l->begin();  
while(it != NULL)  
{  
    if (*it == x) {  
        break;  
    }  
    it = it->next();  
}  
return it;
```

Secuencialidad del programa

```
// Colocar la ventana
```

```
Left    = 0;
```

```
Height = 0;
```

```
Show( );
```

```
// Activar los botones
```

```
Boton1->Enabled = true;
```

```
Boton2->Enabled = true;
```

Secuencialidad del programa: declaración de variables

Var

i : integer;

area : real;

longitud: real;

nombre : string;

edad : integer;

int i;

int x;

int y;

string edad;

double area;

int x, *ptr; // NO

Expresiones Matemáticas

- $c = a * b + c$
 - ... era $(a * b) + c$
 - ... era $a * (b + c)$
- $\text{cars} = \text{lineas} * \text{caracteresPorLinea} + \text{espaciosMargen}$
 - ... ¡era $a *(b + c)!$
- Sin embargo, el código ejecutable generado es exactamente el mismo con paréntesis que sin ellos:
- $((a * b) + c)$
- $a * b + c$

Expresiones Matemáticas (II)

- ¿Donde cortar una línea con una expresión larga?
 - Antes de una subexpresión
 - Antes de un operador
 - Antes de un paréntesis

```
int x = ((a * b + c) / (c * d * d))  
        + (a / (b * c))  
        + ((3.1451927 * b) + d)  
  
;
```


Otras expresiones

- Por ejemplo:

```
while ( *ptrDestino++=*ptrOrigen++ ) ;
```

```
while(it != NULL)
```

```
    it = it->sig == NULL? NULL :
```

```
        it->sig->iterador;
```

Otras expresiones(II)

- Añadiendo unos paréntesis:

```
while ( * ( ptrDestino++ ) = * ( ptrOrigen++ ) ) ;
```

```
while ( it != NULL )
```

```
    it = ( ( it->sig == NULL ) ? NULL :  
          ( it->sig->iterador ) ) ;
```

Otras expresiones(III)

- Versión real legible:

```
while ( *ptrOrigen != 0 ) {  
    *ptrDestino = *ptrOrigen;  
    ++ptrOrigen;  
    ++ptrDestino;  
}
```

Otras expresiones(IV)

- Versión real legible:

```
while (it != NULL)
{
    if (it->sig != NULL)
        it = it->sig->iterador;
    else it = NULL;
}
```

Bloques

- La forma más recomendable de colocar los bloques es marcar el inicio y el fin de un bloque en líneas separadas

```
if (divisor <> 0) then  
begin  
    resultado := dividendo / divisor;  
    writeln(resultado);  
end;
```

Bloques de una sola línea

- Son casi siempre poco recomendables porque introducen confusión. La línea tiende a ser ilegible.
- Recuérdese que los bloques de una sola línea sin marcas de bloque son una posibilidad, no una obligación.

```
if (divisor != 0) {  
    System.out.print(dividendo/divisor);  
}
```

Estructuras de flujo y repetición

- Disposición de las condiciones en un *if()* o en un *while()*.
 - Una *subcondición* por línea, comenzando por el *juntor*. Si es necesario, una condición puede llevar un comentario "a la derecha".
 - Si existen varias subexpresiones condicionales, se pueden indentar respecto a la expresión principal.
 - Elimina la posibilidad de que el bloque de código de una sola línea se indique sin marcas de bloque.

Estructuras de flujo y repetición

```
if ( not(eof(fichEntrada))  
    and (bytesLeidos < 0  
        or bytesLeidos > 1000)  
    and caracter <> FINAL) then  
begin  
    read(fichEntrada, caracter);  
end;
```


Estructuras de flujo y repetición y bloques de una sola línea

- Deben evitarse siempre:

```
if ( not(eof(fichEntrada))  
    and (bytesLeidos < 0  
        or bytesLeidos > 1000)  
    and caracter <> FINAL) then  
read(fichEntrada, caracter); {ilegible}
```

Reglas semánticas de escritura de código

¿Qué significa lo que escribes?
¡Dame una pista!

Nombres de variables según uso

- Devolución de un valor:

```
class Punto {  
    // ...  
  
    public String toString()  
    {  
        String toret = "";  
        toret += "(" + x + ", " + y + ")";  
        return toret;  
    }  
}
```

Nombres de funciones significativos

- Los nombres de métodos deben sugerir qué hacen:

```
int getEdad(Persona) ;
```

```
bool esPalindromo(const string &) ;
```

- Evítesen nombres como los siguientes:

```
int procesar(const FILE *&) ;
```

```
string pasarAuxiliar(const string &) ;
```

Un ejemplo real

El juego del Nim

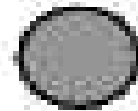
Fila 1



Fila 2



Fila 3



Fila 4



El juego del Nim

- La estrategia del juego se basa en dos métodos, en un programa en Java:

```
class Nim {  
    bool movimiento_chachi(Tablero t) {  
        // ...  
    }  
    bool movimiento_chungo(Tablero t) {  
        // ...  
    }  
}
```

"movimiento_chachi()"

- "*movimiento_chachi()*" prueba si un movimiento le lleva a ganar el juego:

```
bool movimiento_chachi(Tablero t) {  
    // Proponer un movimiento  
    // ...  
    if (movimiento_chungo(t)) {  
        // Realizar el movimiento  
    }  
    // ...  
}
```


"movimiento_chungo()"

- "*movimiento_chungo()*" comprueba si un movimiento es malo ... ¿no?

```
bool movimiento_chungo(Tablero t) {  
    if (t.esVacio()  
        || movimiento_chachi(t))  
    {  
        return false;  
    } else return true;  
}
```

hayMovimientoGanador()

- "*movimiento_chachi()*" puede ser modificado, eliminando "*movimiento_chungo()*":

```
bool hayMovimientoGanador(Tablero t) {  
    // Proponer un movimiento  
    // ...  
    if (!(t.esVacio())  
        && !(hayMovimientoGanador(t))) {  
        // Realizar el movimiento  
    }  
    // ...  
}
```

Programas autodocumentados

Concurso Internacional "C ofuscado"

<http://www.es.ioccc.org/main.html>

```

#define processor x86
#include <stdio.h>
#include <sys/stat.h>
#define l int*
#define F char
    struct stat t;
#define c return
#define I (P+=4,*L(P-4,0))
#define G (signed F)E(*L(P++,0))
#define C(O,D)E (D[B+V(010)/4+O*10])
#define U R[4]=E(V(17)-4),*(l)V(021)=
F      M [99],Q[99],b[9999],*ss,*d=b,*z;
#define O =(n=*(l)V(021),R[4]=E(V(17)+4),n)
#define p(a,b,c) system((sprintf(a,b,k[1]),c)),z
#define
                                g (y/010&7)
#define
                                R (B+13)
#define
                                x86
                                (F*)index\
(ss+V(i
                                ),0100)
#define
                                D(y,n,a,m,i,c
                                )d+=sprintf( d,y,n,a,m,i,c ),(F*\
                                P
                                l
                                B,i,n,a,r,y
                                ,
                                P
                                ;
#define
                                Tr(an,sl,at,or)
                                l an##i(d,sl){ c at? an##i(d,r):or; } \
l an(d,
                                sl){ c \
r=V(014
                                )&63,an##i(d,sl); }
#define add(Ev,Gv) + ...
main (char *ck, char **k) { exit(E((ck?main((z?(stat(M,&t)?
    P+=a+'{'?0:3:execv(M,k),a=G,i=P,y=G&255,sprintf(Q,y/ '@' ...
}

```

```

#include <stdio.h>
int l;int main(int o,char **O,
int I){char c,*D=O[1];if(o>0){
for(l=0;D[l]
];D[l
++]-=10){D[l++]-=120;D[l]-=
110;while(!main(0,O,l))D[l]
+=20; putchar((D[l]+1032)
/20);}putchar(10);}else{
c=o+(D[I]+82)%10-(I>1/2)*
(D[I-1+I]+72)/10-9;D[I]+=I<0?0
:!(o=main(c/10,O,I-1))*((c+999
)%10-(D[I]+92)%10);}return o;}

```

```

r(
) {z
k=0,1
=0,n,x
XQueryPointer(i
,XRootWindow (i,j),&m,
&m,&o,&p,&n,&n,(
ghj)&n),(o
>=s(g)||s(o
)<=0)&&(k=1),
(p>=h||p<=0)&&
(l=1),(e==1)&&(
c=o,d=p,e=0,1)|| (
(k==0&&o-c-(z)(a+y
(a)*.5)!=0)&&(a=o-c
),(l^1==1&&p-d-(z)(
b+y(b)*.5)!=0)&&(b=p-d),a/=f,b/=f
,k=0,l=0);(o >=s(g)||o<=0)&&(a=-a),(
&&(b=-b),c=o,d=p,I(XWarpP
,ointer)(i,None,None,0,0,s
(g),h,(z)(a+y(a)*.5),(int)(
b+y(b)*.5 JJ(float B;int)C,D;
#else/*Egads! something has */
#include<X11/Xlib.h> /*taken a*/
#include<stdio.h> /*huge bite o-*/
#include<stdlib.h> /*ut of the m-*/
#include<time.h> /*ouse pointer!!!*/
#define H(a, b) ((a)&(7<<3*(b)))>>3*(b))
#define G(c,d) ((H(c,d)<<3*(d+1))|(H(c,d+1)<<3*d)|/*
_XSetPointer(display, screen,GREASY|BOUNCY)*/c&~(63<<3*(d)))
#define
s(e) (G(G(G(G(G(e,(z)0),1),2),1),0),1))
typedef int z;float a=0,b=0,c,d,f=1.03;z e
=s(512),g,h,j;
Display/**/*i;
#define y(X)((X>0)-(X<0))
#define x o,p; Window m;
#define ghj unsigned int*
#define I(aa,bb)aa##bb
#define JJ(X\
);return 0;}X
z r();int main
(z X,char**Y){
clock_t q=0;(X
==2)&&(f=atof(Y[1])),(i
=XOpenDisplay(0) )==0)&&(exit(1
),1),j=I(Defa, ultScreen)(i),
g=s(I(Display,
Width)(i,j)-1)
,h=I(DisplayH,
eight)(i,j)-1;
for(;;{(I(clo,
ck)()-q)*100>(
CLOCKS_PER_SEC
))&&(r(),q=clock());}
#include __FILE__
#endif

```

Bibliografía

- Eckel (2000). *Thinking in C++*. Prentice Hall
- Caro, Ramos, Barceló (2002). *Introducción a la programación con orientación a objetos*. Prentice-Hall
- Documentación de Microsoft:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/stg/stg/coding_style_conventions.asp
- Documentación de Sun:
<http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>

Coloquios Abiertos

Programando con estilo

¿Escribo un programa o un jeroglífico?



Jueves 15 de Abril de 2004
Edif. Politécnico
Salón de grados, 18:30h

J. Baltasar García Perez-Schofield

<http://trevinca.ei.uvigo.es/~ca/>

<http://trevinca.ei.uvigo.es/~jgarcia/ca/>