

Re: Object-oriented programming in standard ANSI C

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2006-11/msg02476.html

- *From:* "E. Robert Tisdale" <edwin@xxxxxxxxxxxx>
 - *Date:* Fri, 17 Nov 2006 10:53:58 -0800
-

Thierry Chappuis wrote:

I'm interested in techniques used to program in an object-oriented way using the C ANSI language. I'm studying the GObject library and Laurent Deniau's OOPC framework published on his web site at <http://ldeniau.web.cern.ch/ldeniau/html/oopc/oopc.html>. The approach is very instructive. I know that I could do much of this stuff with e.g. C++, but the intellectual challenge of implementing these concepts with pure ANSI C is relevant to me.

Are you aware of another approaches? Any experience in using such techniques in production code? The use of GObject seems to be well implemented in the GNOME world, but I didn't find much about Laurent Deniau's OOPC. Have you some comments about the strengths and drawbacks of such techniques?

Please find attached a C implementation of Bjarne Stroustrup's famous Shape class.

C is not an object oriented programming language but you can write object oriented programs in C including programs that implement run-time polymorphism. The `fprintf` function on type `*FILE` is an example. You don't need a language translator or a special library.

A program is not an object oriented programming language just because it is written in an object oriented programming language. All the popular object oriented programming languages are procedural programming languages first. An object oriented program must actually use the object oriented features of the language. Object oriented programming is first of all a programming style.

The problem with C is that it does not directly support features such as inheritance and run-time polymorphism. Good C programmers have always used an object oriented programming style

Re: Object-oriented programming in standard ANSI C

but implementing object oriented programs in C is problematic — tedious and error prone. The C++ programming language provides direct support for object oriented programming so it is much easier and more reliable.

```
> cat Point.h
```

```
#ifndef GUARD_Point_h
#define GUARD_Point_h 1

typedef struct Point {
// representation
double X;
double Y;
} Point;
// functions
double
Point_x(const Point* p);
double
Point_y(const Point* p);
Point*
Point_initialize(Point* p, double x, double y);
// constructors
Point
Point_createExplicit(double x, double y);
Point
Point_createDefault(void);
Point*
Point_newExplicit(double x, double y);
Point*
Point_newDefault(void);
// destructors
void
Point_destroy(const Point* p);
void
Point_delete(const Point* p);

#endif // GUARD_Point_h
```

```
> cat Point.c
```

```
// gcc -Wall -std=c99 -pedantic -I. -O2 -c Point.c
```

```
#include<Point.h>
```

```
#include<stdlib.h>
```

```
// functions
```

```
double
```

```
Point_x(const Point* p) {
return p->X; }
```

```
double
```

```
Point_y(const Point* p) {
return p->Y; }
```

```
Point*
```

Re: Object-oriented programming in standard ANSI C

```
Point_initialize(Point* p, double x, double y) {
p->X = x;
p->Y = y;
return p;
}
// constructors
Point
Point_createExplicit(double x, double y) {
Point p;
Point_initialize(&p, x, y);
return p;
}
Point
Point_createDefault(void) {
return Point_createExplicit(0.0, 0.0); }
Point*
Point_newExplicit(double x, double y) {
Point* p = (Point*)malloc(sizeof(Point));
Point_initialize(p, x, y);
return p; }
Point*
Point_newDefault(void) {
return Point_newExplicit(0.0, 0.0); }
// destructors
void
Point_destroy(const Point* p) { }
void
Point_delete(const Point* p) {
Point_destroy(p);
free((void*)p); }
```

```
> cat Color.h
#ifndef GUARD_Color_h
#define GUARD_Color_h 1
typedef struct Color {
// representation
unsigned int R; // red
unsigned int G; // green
unsigned int B; // blue
} Color;
// functions
unsigned int
Color_red(const Color *c);
unsigned int
Color_green(const Color *c);
unsigned int
Color_blue(const Color *c);
Color*
Color_initialize(Color* c,
unsigned int r,
unsigned int g,
```

Re: Object-oriented programming in standard ANSI C

```
unsigned int b);
// constructors
Color
Color_createExplicit(
unsigned int r,
unsigned int g,
unsigned int b);
Color
Color_createDefault(void);
Color*
Color_newExplicit(
unsigned int r,
unsigned int g,
unsigned int b);
Color*
Color_newDefault(void);
// destructor
void
Color_destroy(const Color *c);
void
Color_delete(const Color *c);

#endif // GUARD_Color_h

> cat Color.c
// gcc -Wall -std=c99 -pedantic -I. -O2 -c Color.c

#include<Color.h>
#include<stdlib.h>

// functions
unsigned int
Color_red(const Color *c) {
return c->R; }
unsigned int
Color_green(const Color *c) {
return c->G; }
unsigned int
Color_blue(const Color *c) {
return c->B; }
Color*
Color_initialize(Color* c,
unsigned int r,
unsigned int g,
unsigned int b) {
c->R = r;
c->G = g;
c->B = b;
return c;
}
// constructors
```

Re: Object-oriented programming in standard ANSI C

```
Color
Color_createExplicit(
unsigned int r,
unsigned int g,
unsigned int b) {
Color c;
Color_initialize(&c, r, g, b);
return c; }
Color
Color_createDefault(void) {
return Color_createExplicit(0, 0, 0); }
Color*
Color_newExplicit(
unsigned int r,
unsigned int g,
unsigned int b) {
Color* c = (Color*)malloc(sizeof(Color));
Color_initialize(c, r, g, b);
return c; }
Color*
Color_newDefault(void) {
return Color_newExplicit(0, 0, 0); }
// destructors
void
Color_destroy(const Color *c) { }
void
Color_delete(const Color *c) {
Color_destroy(c);
free((void*)c); }
```

```
> cat Shape.h
#ifndef GUARD_Shape_h
#define GUARD_Shape_h 1

#include<Point.h>
#include<Color.h>

typedef struct Shape {
// representation
const
void* V; // virtual function table pointer
Point P;
Color C;
} Shape;
// functions
const Point*
Shape_point(const Shape* s);
const Color*
Shape_color(const Shape* s);
void
actualShape_draw(const Shape* s);
```

Re: Object-oriented programming in standard ANSI C

```
double
actualShape_area(const Shape* s);
void
Shape_draw(const Shape* s); // virtual function
double
Shape_area(const Shape* s); // virtual function
Shape*
Shape_initialize(Shape* s,
const void* v,
const Point* p,
const Color* c);
// constructors
Shape
Shape_createExplicit(
const Point* p,
const Color* c);
Shape
Shape_createDefault(void);
Shape*
Shape_newExplicit(
const Point* p,
const Color* c);
Shape*
Shape_newDefault(void);
// destructors
void
Shape_destroy(const Shape* s);
void
Shape_delete(const Shape* s);

#endif // GUARD_Shape_h

> cat Shape.c
// gcc -Wall -std=c99 -pedantic -I. -O2 -c Shape.c

#include<stdio.h>
#include<Shape.h>
#include<stdlib.h>

// functions
const Point*
Shape_point(const Shape* s) {
return &(s->P); }
const Color*
Shape_color(const Shape* s) {
return &(s->C); }
void
actualShape_draw(const Shape* s) {
fprintf(stderr, "Shape_draw(const Shape*)\n");
fflush(stderr); }
double
```

Re: Object-oriented programming in standard ANSI C

```
actualShape_area(const Shape* s) {
fprintf(stderr, "Shape_area(const Shape*)\n");
fflush(stderr);
return 0.0; }

typedef struct Shape_vtable_t {
void (*Shape_draw)(const Shape*);
double (*Shape_area)(const Shape*);
} Shape_vtable_t;
static const
Shape_vtable_t
Shape_vtable = {actualShape_draw, actualShape_area};

void
Shape_draw(const Shape* s) { // virtual function
((Shape_vtable_t*)(s->V))->Shape_draw(s); }
double
Shape_area(const Shape* s) { // virtual function
return ((Shape_vtable_t*)(s->V))->Shape_area(s); }
Shape*
Shape_initialize(Shape* s,
const
void* v,
const
Point* p,
const
Color* c) {
s->V = v;
Point_initialize(&(s->P), Point_x(p), Point_y(p));
Color_initialize(&(s->C), Color_red(c), Color_green(c), Color_blue(c));
return s; }
// constructors
Shape
Shape_createExplicitShape(
const Point* p,
const Color* c) {
Shape s;
Shape_initialize(&s, (const void*)&Shape_vtable, p, c);
return s; }
Shape
Shape_createDefault(void) {
Shape s;
Point p = Point_createDefault();
Color c = Color_createDefault();
Shape_initialize(&s, (const void*)&Shape_vtable, &p, &c);
Color_destroy(&c);
Point_destroy(&p);
return s; }
Shape*
Shape_newExplicitShape(
const Point* p,
```

Re: Object-oriented programming in standard ANSI C

```
const Color* c) {
Shape* s = (Shape*)malloc(sizeof(Shape));
Shape_initialize(s, (const void*)&Shape_vtable), p, c);
return s; }
Shape*
Shape_newDefault(void) {
Shape* s = (Shape*)malloc(sizeof(Shape));
Point p = Point_createDefault();
Color c = Color_createDefault();
Shape_initialize(s, (const void*)&Shape_vtable), &p, &c);
Color_destroy(&c);
Point_destroy(&p);
return s; }
// destructors
void
Shape_destroy(const Shape* s) {
Color_destroy(Shape_color(s));
Point_destroy(Shape_point(s));
}
void
Shape_delete(const Shape* s) {
Shape_destroy(s);
free((void*)s); }
```

> cat Circle.h

```
#ifndef GUARD_Circle_h
#define GUARD_Circle_h 1

#include<Shape.h>

typedef struct Circle {
Shape S; // public base class
double R; // radius
} Circle;
// functions
const Shape*
Circle_shape(const Circle* c);
double
Circle_radius(const Circle* c);
void
actualCircle_draw(const Circle* c);
double
actualCircle_area(const Circle* c);
void
Circle_draw(const Circle* c); // virtual function
double
Circle_area(const Circle* c); // virtual function
Circle*
Circle_initialize(Circle* c, const Shape* s, double r);
// constructors
Circle
```

Re: Object-oriented programming in standard ANSI C

```
Circle_createDefault(void);
Circle
Circle_createExplicit(const Shape* s, double r);
Circle*
Circle_newDefault(void);
Circle*
Circle_newExplicit(const Shape* s, double r);
// destructors
void
Circle_destroy(const Circle* c);
void
Circle_delete(const Circle* c);

#endif // GUARD_Circle_h

> cat Circle.c
// gcc -Wall -std=c99 -pedantic -I. -O2 -c Circle.c

#include<math.h>
#include<stdio.h>
#include<Circle.h>
#include<stdlib.h>

// functions
const Shape*
Circle_shape(const Circle* c) {
return &(c->S); }
double
Circle_radius(const Circle* c) {
return c->R; }
void
actualCircle_draw(const Circle* c) {
fprintf(stderr, "Circle_draw(const Circle*)\n");
fflush(stderr); }
double
actualCircle_area(const Circle* c) {
const
double pi = 3.14159265358979323846;
const
double r = Circle_radius(c);
fprintf(stderr, "Circle_area(const Circle*)\n");
fflush(stderr);
return pi*r*r; }

typedef struct Circle_vtable_t {
void (*Circle_draw)(const Circle*);
double (*Circle_area)(const Circle*);
} Circle_vtable_t;
static const
Circle_vtable_t
Circle_vtable = {actualCircle_draw, actualCircle_area};
```

Re: Object-oriented programming in standard ANSI C

```
void
Circle_draw(const Circle* c) { // virtual function
((Circle_vtable_t*)(c->S.V))->Circle_draw(c);
}
double
Circle_area(const Circle* c) { // virtual function
return ((Circle_vtable_t*)(c->S.V))->Circle_area(c);
}

Circle*
Circle_initialize(Circle* c, const Shape* s, double r) {
Shape_initialize(&(c->S),
(void*)&Circle_vtable,
Shape_point(s),
Shape_color(s));
c->R = r;
return c; }

// constructors
Circle
Circle_createExplicit(const Shape* s, double r) {
Circle c;
Circle_initialize(&c, s, r);
return c; }
Circle
Circle_createDefault(void) {
Circle c;
const
Shape s = Shape_createDefault();
Circle_initialize(&c, &s, 0.0);
Shape_destroy(&s);
return c; }
Circle*
Circle_newExplicit(const Shape* s, double r) {
Circle* c = (Circle*)malloc(sizeof(Circle));
Circle_initialize(c, s, r);
return c; }
Circle*
Circle_newDefault(void) {
Circle* c = (Circle*)malloc(sizeof(Circle));
const
Shape s = Shape_createDefault();
Circle_initialize(c, &s, 0.0);
Shape_destroy(&s);
return c; }
// destructors
void
Circle_destroy(const Circle* c) {
Shape_destroy(Circle_shape(c));
}
void
```

Re: Object-oriented programming in standard ANSI C

```
Circle_delete(const Circle* c) {  
Shape_destroy(Circle_shape(c));  
free((void*)c); }
```

> cat main.c

```
// gcc -Wall -std=c99 -pedantic -I. -O2 -o main main.c Circle.o Shape.o Color.o Point.o
```

```
#include<stdio.h>  
#include<Circle.h>
```

```
int  
main(int argc, char* argv[]) {  
const  
Shape s = Shape_createDefault();  
const  
Circle c = Circle_createExplicit(&s, 2.0);  
Shape_draw((const Shape*)&c);  
fprintf(stdout, "%g = radius\t %g = area\n",  
Circle_radius(&c), Shape_area((const Shape*)&c));  
return 0;  
}
```

> cat Makefile

```
CC=gcc  
DEFINES=  
INCLUDE=-I.  
OPTIONS=-Wall -std=c99 -pedantic -O2  
LIBRARY=  
OBJECTS=Point.o Color.o Shape.o Circle.o  
SOURCES=Point.c Color.c Shape.c Circle.c  
HEADERS=Point.h Color.h Shape.h Circle.h  
library=  
COMPILE=$(CC) $(DEFINES) $(INCLUDE) $(LIBRARY) $(OPTIONS)
```

```
main: $(HEADERS) $(OBJECTS) main.c  
$(COMPILE) -o main main.c $(OBJECTS) $(library)
```

```
Point.o: Point.h Point.c  
$(COMPILE) -c Point.c
```

```
Color.o: Color.h Color.c  
$(COMPILE) -c Color.c
```

```
Shape.o: Shape.h Shape.c  
$(COMPILE) -c Shape.c
```

```
Circle.o: Circle.h Circle.c  
$(COMPILE) -c Circle.c
```

```
clean:  
rm -f main $(OBJECTS)
```