Software Construction Tips

CSE 40166 Computer Graphics (Fall 2009)

Multiple Source Files

- Separate ***interface*** and ***implementation***
    - **Interface:** minimal set of methods required to use library.
    - **Implementation:** code that actually implements the specified interface.
- In C/C++ split interface and implementation into header file and source.
    - **Header:** function prototypes, type definitions, externs, constants.
    - **Source:** function definitions, global variables.

Example: Color utility

- Simple color module that allows you to:
    - Grab colors by name:
        - `ColorTable[COLOR_RED];`
    - Get a random color
        - `random_color();`
    - Rotate through colors:
        - `rotate_color();`

Example: color.h

```
#ifndef __COLOR_H__                // Prevent multiple includes
#define __COLOR_H__

enum COLOR_TABLE_INDEX {           // Enumeration constants
    COLOR_RED = 0,
    COLOR_GREEN,
    COLOR_BLUE,
    ...
    COLOR_UNKNOWN
};

extern GLfloat ColorTable[][3];    // External global variable

GLfloat *rotate_color();           // Function prototypes
GLfloat *random_color();

#endif
```

Example: color.cc

```
#include "color.h"

GLfloat ColorTable[][3] = {
    { 1.0, 0.0, 0.0 },
    { 0.0, 1.0, 0.0 },
    { 0.0, 0.0, 1.0 },
    ...
};
GLfloat *
rotate_color()
{
    static int index = COLOR_RED;
    index = (index + 1) % COLOR_WHITE;
    return ColorTable[index];
}
GLfloat *
random_color()
{
    return ColorTable[rand() % COLOR_UNKNOWN];
}
```

Multiple Objects, Single Interface

- Sometimes you have a group of objects with similar properties or interfaces:
  - **PGM 5:** multiple scene objects like control points, sphere, cube, etc.
    - Location
    - Orientation
    - Scale
    - Color
- One way to handle this is provide a single struct, record the type, and then do a switch statement

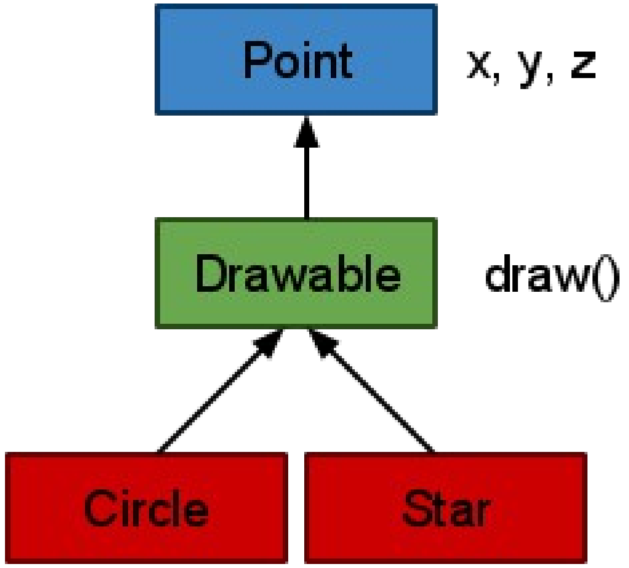
Annotated Structs

```
typedef struct  {
    double x;
    double y;
    double z;
    int         type;
} Drawable;

enum DRAWABLES {
    DRAWABLE_POINT,
    DRAWABLE_SPHERE
};
```

```
Drawable d = { 0.0, 0.0, 0.0,
               DRAWABLE_POINT };

switch (d.type) {
    case DRAWABLE_POINT:
        draw_point();
        break;
    case DRAWABLE_SPHERE:
        draw_sphere();
        break;
}
```

Object Oriented Design

- Create a series of classes using ***inheritance***.
  - Every object can be represented as a *point* (has x, y, z).
  - Every drawable object has a *draw* method.
  - Circles, and stars are drawable objects.



Example: point.h

```
#ifndef __POINT_H__
#define __POINT_H__

class Point
{
public:
    Point(const double pX = 0.0, const double pY = 0.0, const 
double pZ = 0.0)
        : x(pX), y(pY), z(pZ)
    {}

    virtual ~Point()
    {}

    double x;
    double y;
    double z;
};

#endif
```

Example: drawable.h

```cpp
#ifndef __DRAWABLE_H__
#define __DRAWABLE_H__

#include "color.h"
#include "point.h"

class Drawable : public Point
{
public:
    Drawable(const double pX = 0.0, const double pY = 0.0, const double pZ = 0.0)
        : Point(pX, pY, pZ)
    {
        set_color(random_color());
    }

    virtual ~Drawable() {};

    virtual void draw() = 0;
    virtual void render() = 0;

    void set_color(GLfloat pColor[3])
    {
        color[0] = pColor[0]; color[1] = pColor[1]; color[2] = pColor[2];
    }

    GLfloat color[3];
};

#endif
```

Example: drawable_circle.h

```
#ifndef __DRAWABLE_CIRCLE_H__
#define __DRAWABLE_CIRCLE_H__

#include "drawable.h"

class Circle : public Drawable
{
public:
    Circle(const double pX = 0.0, const double pY = 0.0, const
double pZ = 0.0, const double pRadius = 10.0);

    virtual void draw();
    virtual void render();

    static int DisplayListId;

    double radius;
};

#endif
```

Example: main.cc

Drawing a list of objects is now easy:

```
void
display()
{
    glClear(GL_COLOR_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    list<Drawable*>::iterator it;
    for (it = Drawables.begin(); it != Drawables.end(); it++)
        (*it)->draw();

    glutSwapBuffers();
}
```

Compiling multiple files with Make

target-output*: *input-dependencies
    command to run on input to generate output target

```
color.o: color.cc color.h

drawable_circle.o: drawable_circle.cc drawable_circle.h drawable.h color.h
point.h utils.h

main.o: main.cc drawable.h color.h point.h drawable_circle.h drawable_star.h
drawable_initials.h
```

Anytime timestamp for dependency is newer than target, then re-run command to generate target.

Make tricks

- Force update by touching a file:

```
$ touch drawable.h
```

- Automatically generate dependencies using gcc

```
$ g++ -MM *.cc >> Makefile
```