C++ for C programmers

Duncan Mac-Vicar P. <dmacvicar@suse.de>
Team Lead, YaST, Germany
Why?

- Object oriented syntax, with C compatibility
- Language of choice of various projects
- Various flavors to choose from
Object oriented C patterns
Reviewing C patterns

#include <stdlib.h>

typedef struct _Person
{
    char *name;
    int age;
} Person;

int main()
{
    p = (Person *) malloc(sizeof(Person));
    p->age = 36;
    return 0;
}
Objects

• Imagine for a second we are not talking about Persons, but Cars
• We want the car go faster

car->fuelflow++;
car->.... ?
Interfaces
/* person.h */

typedef struct _Person
{
    char *name;
    int age;
} Person;

Person * person_create();
void person_destroy( Person *p );
void person_set_name( Person *p, const char *name );
void person_set_age( Person *p, int age );

#endif
Which can be now used...

```c
int main()
{
    Person *p;
    p = person_create();
    person_set_age(p, 30);
    person_destroy(p);
    return 0;
}
```
Which can be now used...

```c
int main()
{
    Person *p;
    p = person_create();
    person_set_age(p, 30);
    return 0;
}
```
Which can be now used...

```c
int main()
{
    Person *p;
    p = person_create();
    person_set_age(p, 30);
    return 0;
}
```

No knowledge about internals
methods

\texttt{person\_set\_age(p, 30)};

seems to be out of place... we already have the object where we want to perform \texttt{person\_set\_age} on.

(actually, prefixing it with \texttt{person\_} is a hack to avoid collision with other structure's methods and to make it readable ) Why not?:

\texttt{p->set\_age(30)};
Destroying objects

```c
int main()
{
    Person *p;
    p = person_create();
    person_set_age(p, 30);
    return 0;
}
```

Destroy not called  (expected)
Destroying objects

```c
int main()
{
    Person p;
    person_init(&p);
    person_set_age(&p, 30);
    return 0;
}
```

Destroy not called (expected?)
Hierarchies

All employees are persons. However not every person is an employee.

```c
typedef struct _Person
{
   char *name;
   int age;
} Person;

typedef struct _Employee
{
   Person *person;
   int salary;
} Employee;

Employee *e;
...
person_set_age(e, 30);
person_set_age(e->person, 30);
```
Object Oriented concepts and C++
class Light
{
    public:
        void on();
        void off();
        void brighten();
        void dim();
    private:
        int _power;
};

Light light;
light.on();
light._power = 10;
Composition (A has B)

class Engine;

class Car
{
    public:
        Engine * engine();
    private:
        Engine *_engine;
};

Engine * Car::engine()
{
    return _engine;
}
Inheritance ( A is B )

class Shape
{
    public:
        void draw();
        void erase();
        void move();
        Color getColor();
        void setColor( Color color );
};

class Circle : public Shape
{
    public:
        // has all Shape interface
        int radio();
};

Shape *a = new Shape();
Shape *b = new Shape();
Circle *c = new Circle();
a->draw();
c->draw();
c->radio();
b->radio();
b = a; // OK
b = c; // OK
c = a; // wrong!
Overriding functions

class Circle : public Shape
{
    // overrides Shape's
    virtual draw();
};

Shape *a;
Shape *b;
Circle *c;
...
a->draw()  // Shape *, calls Shape's draw
b = c;
b->draw()  // Shape *, calls Circle's draw!!
c->draw()  // Circle *, calls Circle's draw
Runtime polymorphism

BirdController
\texttt{reLocate()}\hspace{1cm} What happens when \texttt{move()} is called?

Bird
\texttt{move()}

Thermostat
\texttt{lowerTemperature()}

Controls

Cooling System
\texttt{cool()}

Goose
\texttt{move()}

Air Conditioner
\texttt{cool()}

Heat Pump
\texttt{cool()}\hspace{1cm} \texttt{heat()}

Penguin
\texttt{move()}
More syntax after the concepts
Methods / inline methods

class Shape
{
    void draw();
};

void Shape::draw();
{
    // blah
}

class Shape
{
    void draw()
    {
        // blah
    }
};
void Circle::doNiceThings();
{
    // the same
    _radio = 10;
    this->_radio = 10;

    // doSomething( Shape * ) or
    // doSomething( Circle * )
    doSomething(this);
}
Constructors

```cpp
typedef int Color;

class Shape
{
public:
    Shape();
    Shape( Color c );

private:
    Color _color;
};

Shape s;
Shape s(10);
s.draw();

Shape *s = new Shape();
Shape *s = new Shape(10);
s->draw();
```
Calling superclass constructors

Circle::Circle(Color c, int radio)
  : Shape(c), _radio(radio)
{
  // more stuff here
}

Destructors

```cpp
class Shape
{
public:
    Shape();
    ~Shape();
};
Shape::~Shape()
{
    // cleanup
}
Shape *s = new Shape();
delete s; // destructor called

// plain block
{
    Shape s;
    // destructor called
}
```
Compile time restrictions: const method

class Shape
{
  public:
    int color() const
    {
      return _color;
    }
  void setColor( int c )
  {
    _color = c;
  }
  private:
    int _color;
};
Default arguments

class Shape
{
public:
    int color() const:
    void setColor( int c = 0 /* black */ );
private:
    int _color;
};

void Shape::setColor( int c )
{ ... }

s->setColor(2);
s->setColor();
Templates
rethinking the linked list...
Templates

template <class T>
class mypair {
    T values [2];
    public:
        mypair (T first, T second)
        {
            values[0]=first; values[1]=second;
        }
};

mypair<int> myobject (115, 36);
Languages, Stacks and Frameworks
STL

http://www.boost.org/doc/libs

boost

STL

c++

smart ptrs, threads, functors, testing, algorithms, etc

strings, containers, i/o streams

language
http://doc.trolltech.com/
http://api.kde.org/

KDE

Qt

c++

Desktop, hardware, multimedia, Application,

Strings, containers, i/o streams, GUI, network, testing, scripting, signals, ipc, etc

language
Forget const char *

```cpp
string a("abcd efg");
string b("xyz ijk");
string c;

cout << a << " " << b << endl;  // Output: abcd efg xyz
                             // ijk

cout << "String empty: " << c.empty() << endl;  // String empty: 1
// Is string empty? Yes it is empty. (TRUE)
c = a + b;                         // concatenation
    // abcd efgxyz ijk

cout << "String length: " << c.length() << endl;  // String length: 15

cout << "String size: " << c.size() << endl;     // String size: 15

cout << "String capacity: " << c.capacity() << endl;  // String capacity: 15

cout << "String empty: " << c.empty() << endl;   // String empty: 0
// Is string empty? No it is NOT empty. (FALSE)
string d = c;
cout << d << endl;                 // abcd efgxyz ijk
```
How to compile

- Use `g++ -o myprog myprog.cpp`
- Or write a Makefile
- Or write a CmakeLists.txt

PROJECT(foo)
ADD_EXECUTABLE(foo foo.cpp)
Homework #1

Look at Qt toolkit class hierarchy
http://doc.trolltech.com/extras/qt41-class-chart.pdf

Find the parallel between inheritance concepts explained and the fact that every widget takes a QWidget * as a parent.

Read

How to design good APIs
Homework #2

Read
http://developer.kde.org/~wheeler/cpp-pitfalls.html

And learn about the difference between const method, const parameters, const return values.
Homework #3

• Find out a C library written in a more or less object oriented way.
• Wrap it in a C++ class hierarchy (no need to cover the full API)
  • Creating C structs in constructors
  • Destroying the data in the class destructors
  • Wrapping the methods as class methods
• Suggestions: curl easy API, SDL, dbus
Thanks a lot! Questions?
Dankeschön! Fragen?
Muchas gracias! Preguntas?

Join us!
yast-devel@opensuse.org
http://opensuse.org/YaST
General Disclaimer
This document is not to be construed as a promise by any participating company to
develop, deliver, or market a product. Novell, Inc., makes no representations or warranties
with respect to the contents of this document, and specifically disclaims any express or
implied warranties of merchantability or fitness for any particular purpose. Further, Novell,
Inc., reserves the right to revise this document and to make changes to its content, at any
time, without obligation to notify any person or entity of such revisions or changes. All Novell
marks referenced in this presentation are trademarks or registered trademarks of Novell,
Inc. in the United States and other countries. All third-party trademarks are the property of
their respective owners.

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike
2.5 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-
sa/2.5/.

For other licenses contact author.