

11/19/2009

(1)

STL Algorithms

[power of algorithms
predicate functions
function objects]

Derivation

↳ Evolution of classes
↳ derived classes

- — Containers
- — Iterators
- Algorithms \Rightarrow generic use of functions using iterators
- function objects

- count
- find
- remove
- replace

work with iterators
single values

Algorithms that work with
 PREDICATE functions

```

bool isPrime (int);
  list<int> db; int count=0;
  list<int>::iterator p = db.begin();
  while ( p != db.end() )
  {
    if (isPrime(*p)) count++;
    p++;
  }
  unary PREDICATE (1 parameter)
  returns "bool"
  
```

3

STL Algorithms for predicates

count_if, find_if, remove_if,
replace_if

are used with iterators / predicate
function

```
bool isPrime(int);
```

```
list<int> db;
```

```
cout << "Num of primes:"
```

```
<< count_if(db.begin(),  
            db.end(),
```

```
            isPrime);
```

name the predicate

To find 1st prime

```
list<int>::iterator p;
```

```
p = find_if (db.begin(), db.end(),  
            isPrime);
```

```
if (p == db.end())  
    cout << "No prime #";
```

else

```
    cout << "First prime " << *p;
```

Removing Prime #s

```
p = remove_if (db.begin(), db.end(),  
              isPrime);
```

```
db.erase (p, db.end());
```

```
replace_if (db.be.
```

Replacing all prime#s with 100

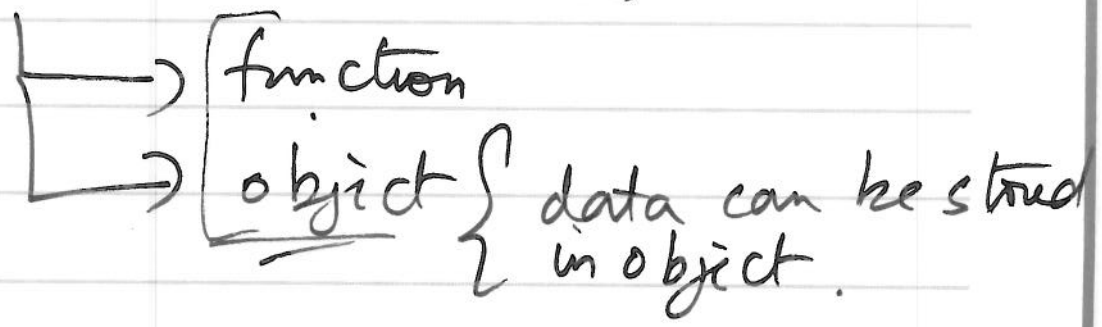
```
replace_if (db.begin(), db.end(),
           isPrime, 100);
```

bool greater_than_50 (int x)

{

doesn't generalize well.

FUNCTION OBJECTS / FUNCTOR



How To Define Function Object

6

```
class greater_than_x
{
public:
    greater_than_x (int v) { value = v; }
    bool operator () (int x)
    {
        if (x > value) return true;
        return false;
    }
private:
    int value;
};
```

function
call
operator

Overload the call operator

```
greater_than_x fun(20);
```

```
cout cout << count_if (db.begin(),
                        db.end(),
                        fun);
```

```
cout << count_if (db.begin(),
                  db.end(),
                  greater_than_x
                  fun(50));
```

replace_if (db.begin(), db.end(),
greater_than_x(1000), ϕ);

Assignment #10

NO LATE SUBMISSION

CLASS EVOLUTION

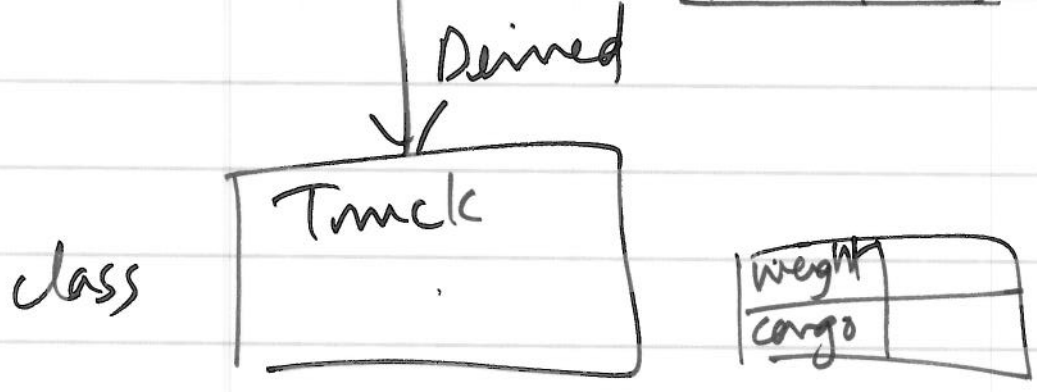
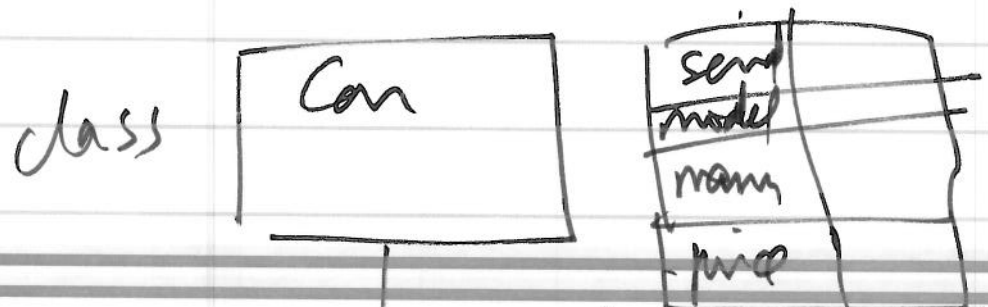
CLASSES : \implies Components

{ interface
implementation

{ EVOLUTION is EXPENSIVE *
" is INEVITABLE * }

O/O Programming

- ENCAPSULATION
- INHERITANCE
 - Derivation of classes.



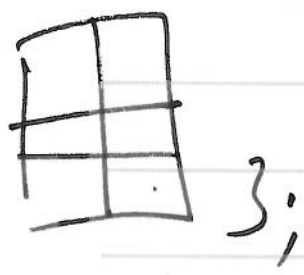
DERIVATION is C++
 Answer to Evolution

class Car

```

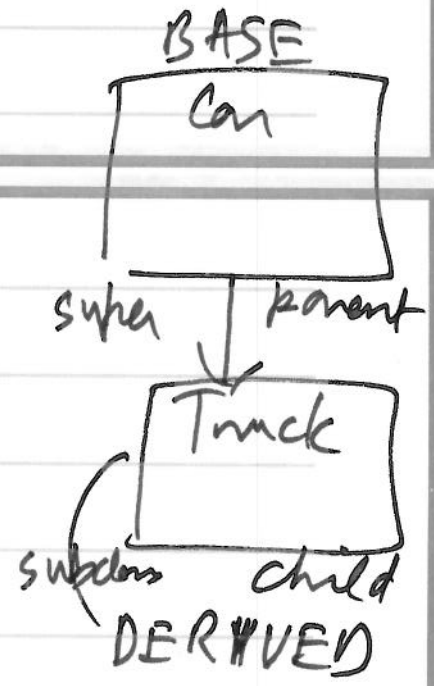
{ public:
    get Serial
    set Serial
    get Man
    set Man
private: display()

```



```

string serial, manufacturer;
int price;
```



class Truck : public Car

{

};

```

Truck X;
X. getSerial()
```

DERIVATION

L Focus on differences

L Truck has some new data
 { cargo
 } tonnage

→ Truck has some new behavior
 { member functions }

class Truck public Car

{ public:

new functions { int getTonnage () { return tonnage; }
 void setTonnage (int t);
 string getCargo () { return cargo; }
 void setCargo (string);
 void display ();

OVERRIDING

private:

int tonnage;
 string cargo;

};

```

void Truck:: setCargo (string c)
{
  //
  //
  //
}

```

```

void Truck:: setTonnage (int t)
{
  //
  //
  //
}

```

```

}
void Truck:: display () // specific
                        to Truck
{
  cout Car:: display ();
  cout << "Tonnage " << tonnage
  << " Cargo " << cargo;
}

```

Truck X;

Car Y;

Y.setSerial("111");

X.setSerial("222"); //inheritance

~~Y~~.Y.display(); //Car::display

X.display(); //Truck::display

X.Car::display();