CS100 Recitation 7

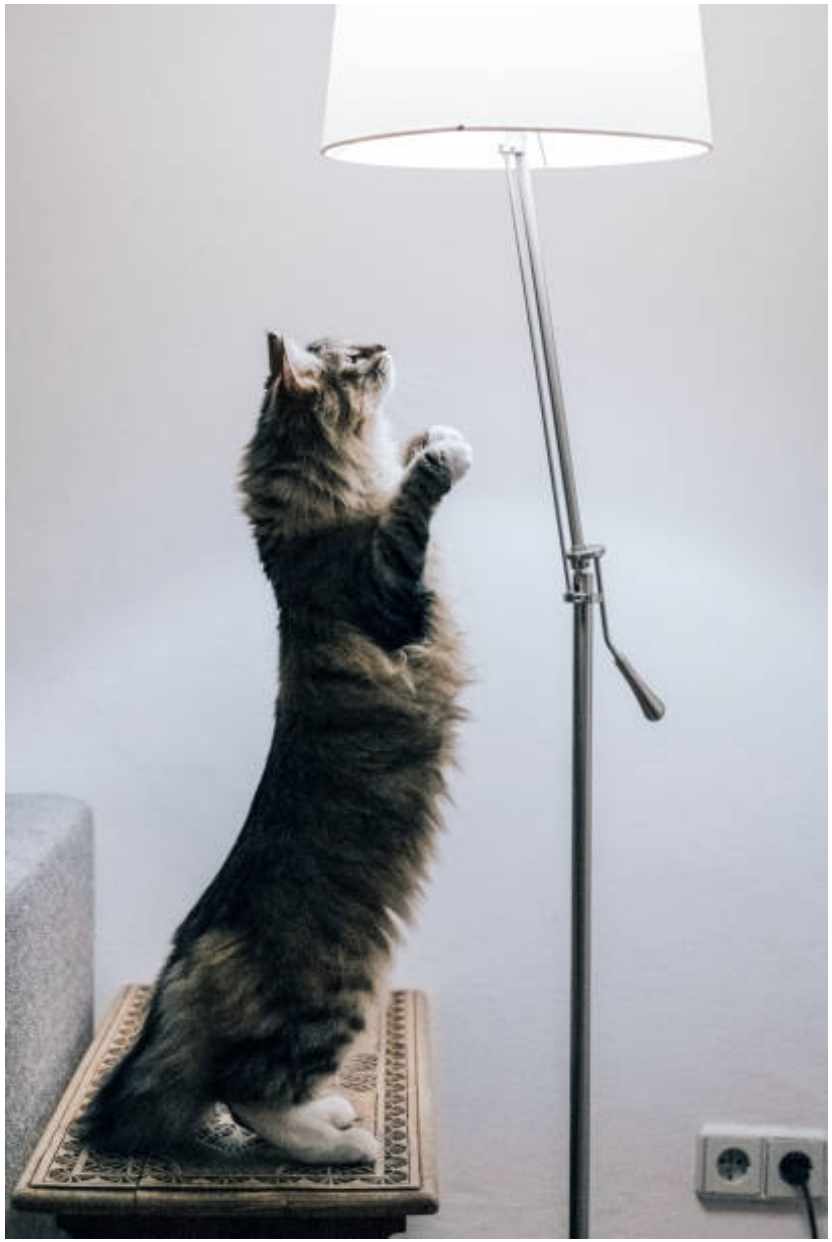
GKxx

April 4, 2022

Drawbacks of a Simple `struct`

Take the `Linked_list` as an example:

- Users can directly access and modify the structure of the list, **without letting the list know!**
- Even though methods of 'create' and 'destroy' are provided, memory management is still a problem because users may forget to call them (or fail to call them correctly).
- The name of every function starts with '`linked_list`', which is lengthy and inconvenient.



Separate Implementation Details and Interfaces

```
struct Point2d {
 private:
  // implementation details
  double x, y;
 public:
  // interfaces
  void set_x(double new_x)
    { x = new_x; }
  void set_y(double new_y)
    { y = new_y; }
  double get_x()
    { return x; }
  double get_y()
    { return y; }
};
```

Access modifiers:

- `private`: Only the code inside the class (or in a `friend`) can access.
- `public`: Everyone can access.
- `protected`: Only the code inside the class or in a subclass, or in a `friend` can access.

Separate Implementation Details and Interfaces

- Implementation details should be invisible to others.
- Interafces are defined for others to use.

```
// In C++, we can directly use the name Point2d without the
    struct keyword.
// The weird typedefs are not needed, either.
Point2d p;
p.x = 4.2;  // Error!
p.set_x(4.2);
p.set_y(3.5);
std::cout << "(" << p.get_x() << ", "
          << p.get_y() << ")" << std::endl;
```

`class` or `struct` ?

In C++, **the only differences** between `class` and `struct` are

- Default access level for a `class` is `private`, while for a `struct` is `public`.

```
class Point {
  double x, y; // private here
  // Other members.
};
```

```
struct Point {
  double x, y; // public here
  // Other members.
};
```

- Default inheritance protection level for a `class` is `private`, while for a `struct` is `public`.

const Member Functions

```
void print_point(const Point2d &p) {
  std::cout << "(" << p.get_x() << ", "
            << p.get_y() << ")" << std::endl;
}
```

- The parameter should be declared as const reference, since it is not modified.
- However, the code above won't compile.
- We need to specify what we can do on const objects.

const Member functions

```
struct Point2d {
 private:
  double x, y;
 public:
  void set_x(double new_x)
    { x = new_x; }
  void set_y(double new_y)
    { y = new_y; }
  double get_x() const
    { return x; }
  double get_y() const
    { return y; }
};
```

- On a non-const object, both const members and non-const members can be called.
- On a const object, only the const members can be called.
- A const member function should not modify the data members.

The this Pointer

Inside a member function, when we refer to the name of a member, we are in fact referring to it through the this pointer.

```
class Point2d {
  double x, y;
 public:
  void set_x(double new_x) {
    this->x = new_x;    // equivalent to x = new_x;
  }
  // Other members.
};
```

- this is a pointer that points to the object itself. For example, in 'class Point2d', this is of type Point2d *.

Name Lookup in `class`

An exception to the name lookup rule:

- Inside a `class` , all the class members are visible, no matter they are before or after the usage.

```
class Point2d {
 public:
  void set_x(double new_x)
    { x = new_x; }       // OK: The member 'x' is visible here.
  void set_y(double new_y)
    { y = new_y; }
  double get_x() const
    { return x; }
  double get_y() const
    { return y; }
 private:
  double x, y;
};
```

Defining Member Functions outside the `class`

A member function can be defined outside the `class` definition, **but must be declared inside the class**.

```
class Point2d {
 public:
  void set_x(double new_x) {
    x = new_x;
  }
  void set_y(double new_y) {
    y = new_y;
  }
  double get_x() const;
  double get_y() const;
 private:
  double x, y;
};
```

- Use `operator::` to refer to a name in the class scope.

```
double Point2d::get_x() const {
  return x;
}
double Point2d::get_y() const {
  return y;
}
```

- The `const` keyword, if needed, must appear at both declaration and definition. It is a part of the function type.

Reference to the Object itself

```
class Point2d {
 public:
  Point2d &set_x(double new_x) {
    x = new_x;
    return *this;
  }
  Point2d &set_y(double new_y) {
    y = new_y;
    return *this;
  }
  // Other members.
};
```

- `set_x` and `set_y` returns a reference to the object itself (which is an lvalue) by `return *this;`
- Then we can do:

```
p.set_x(4.2).set_y(3.5);
```