Class

Point

```
typedef struct point {
    float x;
    float y;
} Point;
Point a;
a.x = 1;a.y = 2;
void print(Point* p)
{
    printf("%d %d\n",p->x,p->y);
}
print(&a);
```

move (dx,dy)?

```
void move(Point* p,int dx, int dy)
{
    p->x += dx;
    p->y += dy;
}
```

Prototypes

```
typedef struct point {
    float x;
    float y;
} Point;
void print(Point* p);
void move(Point* p,int dx, int dy);
```

Usage

```
Point a;
Point b;
a.x = b.x = 1;
a.y = b.y = 1;
move(&a,2,2);
print(&a);
print(&b);
```

Quiz I

- Why all these functions take a pointer to a Point as the first parameter?

Put functions inside

```
struct Point {
    void init(int x,int y);
    void move(int dx,int dy);
    void print();
    int x;
    int y;
};
```

implementations

```
void Point::init(int ix, int iy)
{
    x = ix; y = iy;
}
void Point::move(int dx,int dy)
{
    x+= dx; y+= dy;
}
void Point::print()
{
    cout << x << ' ' << y << endl;
}
```

`::` resolver

- <Class Name>::<function name>
- ::<function name>

```
void S::f() {
   ::f(); // Would be recursive otherwise!
   ::a++; // Select the global a
   a--;   // The a at class scope
}
```

To call a functions in a class

```
Point p;
p.move(10,10);
```

- There is a relationship with the function be called and the variable to call it.
- The function itself knows it is doing something with that variable.

`this` : the hidden parameter

- `this` is a hidden parameter for all member functions, with the type of the class

```
void Point::move(int dx, int dy);
```

➔ (can be recognized as)

```
void Point::move(Point *this, int dx, int dy);
```

- To call the function, you must specify a variable

```
Point p;
p.x=p.y=0;
p.move(10,10);
```

➔ (can be recognized as)

```
Point::move(&p,10,10);
```

this: the pointer to the variable

- Example: this.cpp
- `this` is a natural local variable of all structs member functions that you can not define, but can use it directly.
- Inside member functions, you can use `this` as the pointer to the variable that calls the function.
- Example: Integer.h, Integer.cpp

- To call other member functions within a member function, you do NOT specify the variable:

```
void Point::move_and_print(int dx, int dy) {
    move(dx,dy);
    print();
}
```

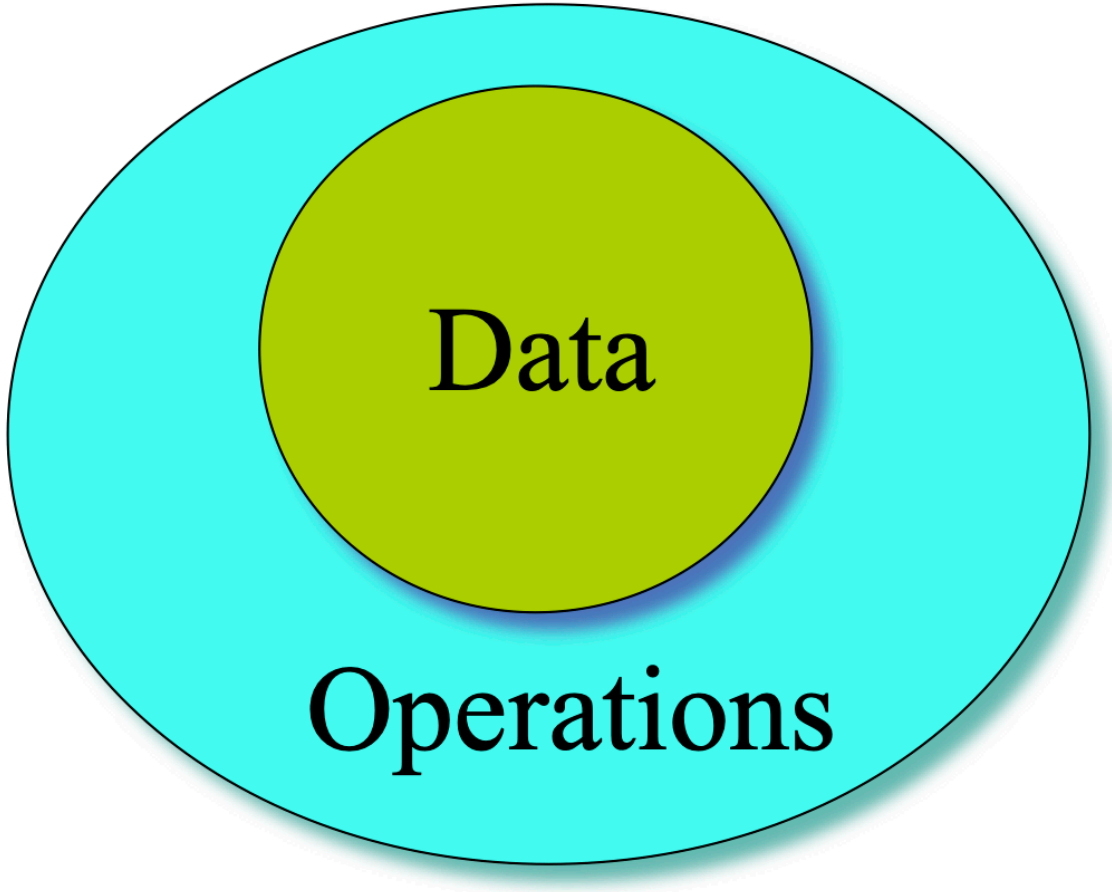
Quiz II

- 有人喜欢在敲成员函数代码的时候，需要调用其他成员函数时，先敲`this->`，请问这样做的好处是什么？

```
void Point::move_and_print(int dx, int dy) {
    this->move(dx,dy);
    this->print();
}
```

Object = Attributes + Services

- Data: the properties or status
- Operations: the functions



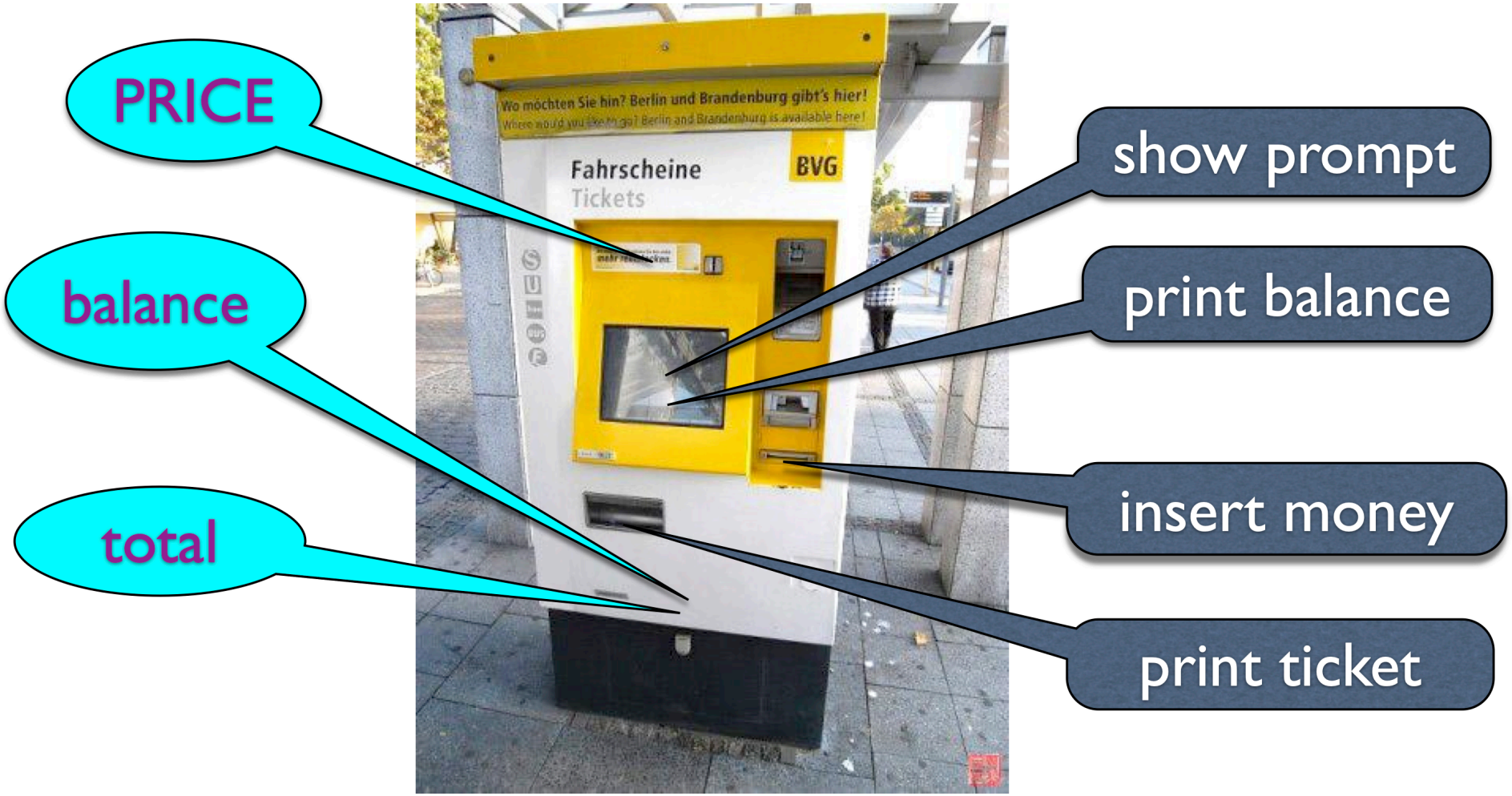
Ticket Machine

- Ticket machines print a ticket when a customer inserts the correct money for their fare.
- Our ticket machines work by customers ‘inserting’ money into them, and then requesting a ticket to be printed. A machine keeps a running total of the amount of money it has collected throughout its operation.

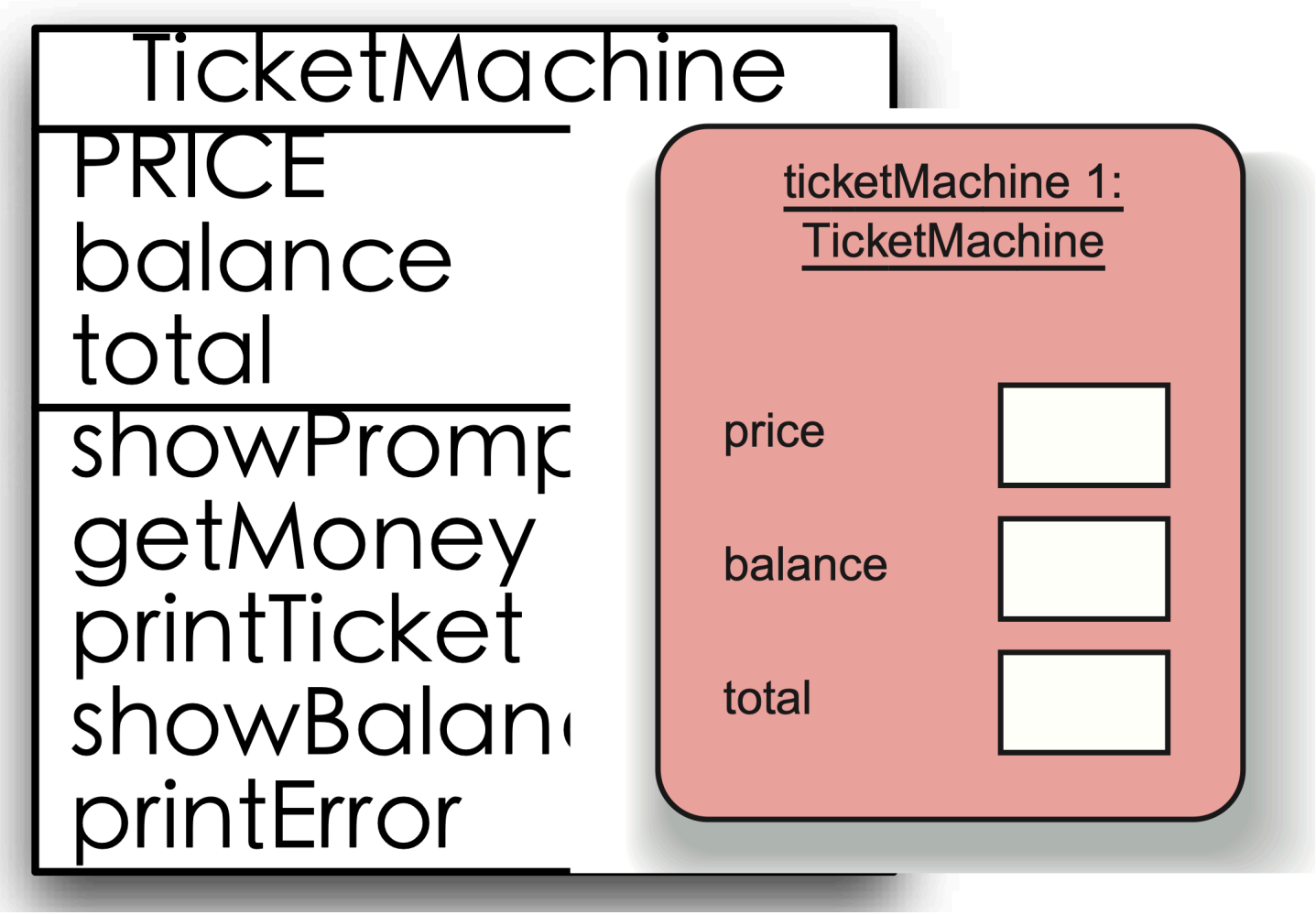
Procedure-Oriented

- Step to the machine
- Insert money into the machine
- The machine prints a ticket
- Take the ticket and leave
- We make a program simulates the procedure of buying tickets. It works. But there is no such machine.
- There's nothing left for the further development

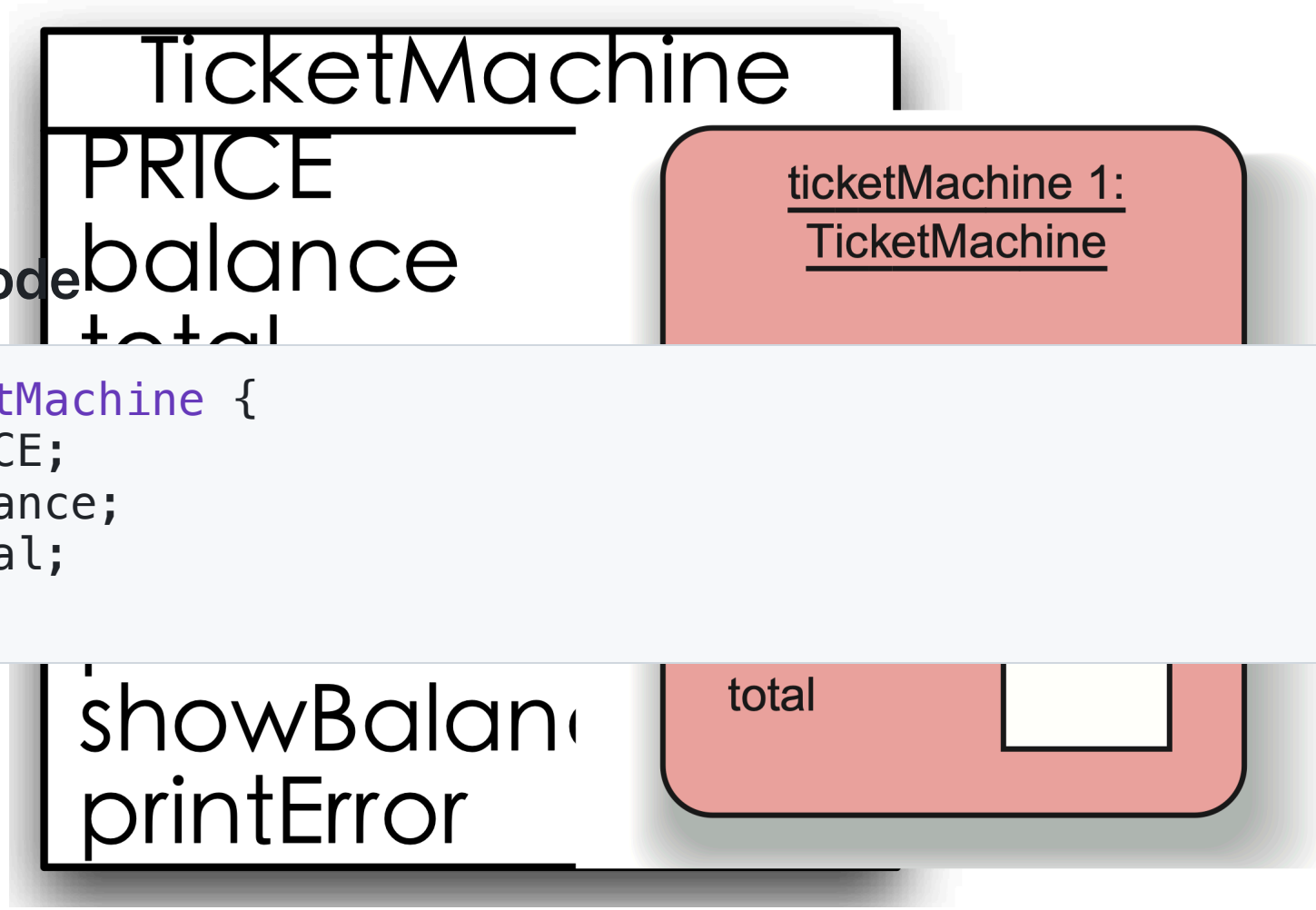
Something is there



Something is here



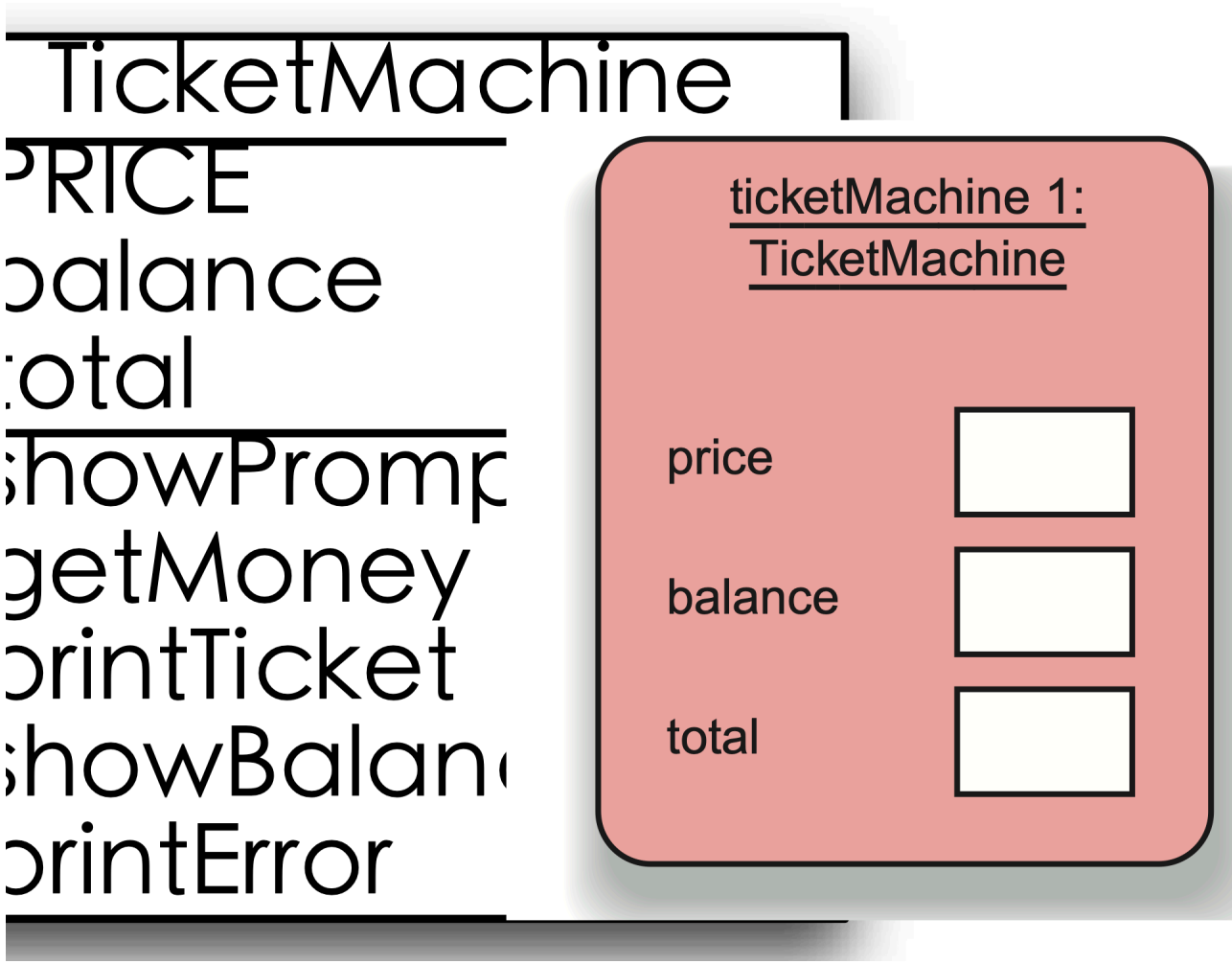
Turn it into code




```
class TicketMachine {
    int PRICE;
    int balance;
    int total;
};
```

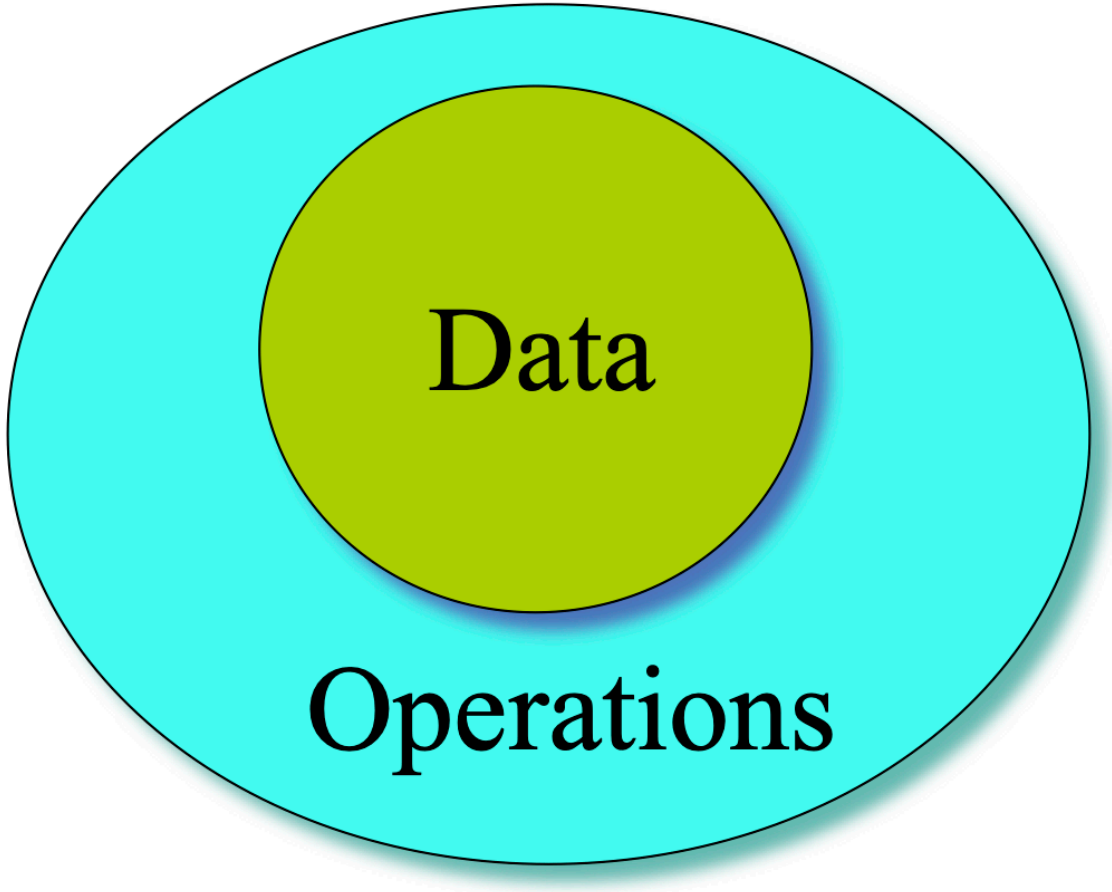
Turn it into code(cn'td)

```
class TicketMachine {
    void showPrompt();
    void getMoney();
    void printTicket();
    void showBalance();
    void printError();
    int PRICE;
    int balance;
    int total;
};
```



Object = Attributes + Services

- Data: the properties or status
- Operations: the functions



Quiz III

- 在教室里找一个对象，写出它的属性和操作

Declaration and Definition of a Class

- Every object has a type. by Alan Kay
- The class declaration, along with the prototype of the member functions should be put into a header file
- The definition of the member functions should be put into another source file

Definition of a class

- In C++, separated `.h` and `.cpp` files are used to define one class.
- Class declaration and prototypes in that class are in the header file (`.h`).
- All the bodies of these functions are in the source file (`.cpp`).

compile unit

- The compiler sees only one .cpp file, and generates .obj file
- The linker links all .obj into one executable file
- To provide information about functions in other .cpp files, use .h

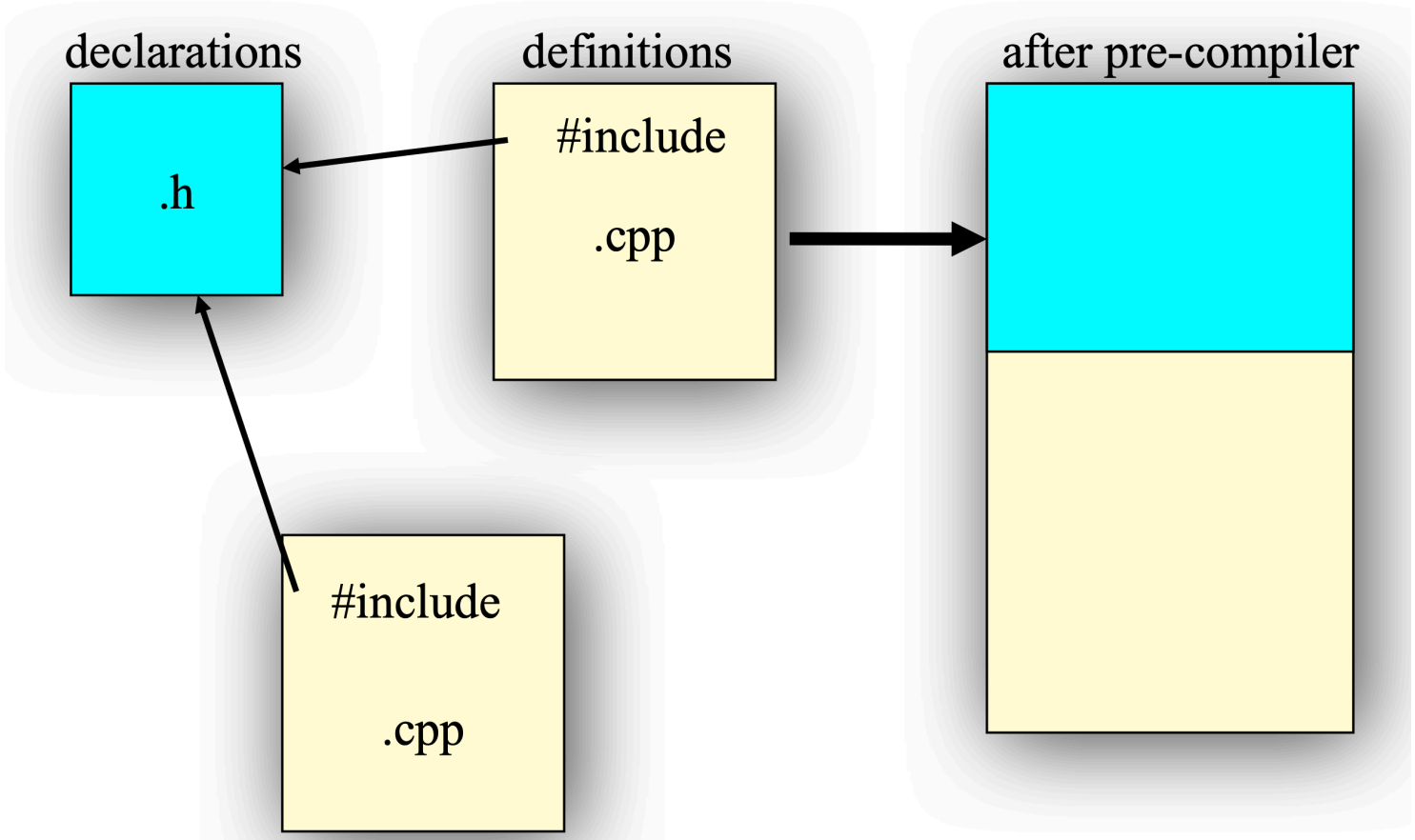
The header files

- If a function is declared in a header file, you must include the header file everywhere the function is used and where the function is defined.
- If a class is declared in a header file, you must include the header file everywhere the class is used and where class member functions are defined.

Header = interface

- The header is a contract between you and the user of your code.
- The compiler enforces the contract by requiring you to declare all structures and functions before they are used.

Structure of C++ program



Declarations vs. Definitions

- A .cpp file is a compile unit
- Only declarations are allowed to be in .h
    - extern variables
    - function prototypes
    - class/struct declaration

`#include` Review

- `#include` is to insert the included file into the .cpp file at where the `#include` statement is.
  - `#include "xx.h"` : search in the current directory firstly, then the directories declared somewhere
  - `#include <xx.h>` : search in the specified directories
  - `#include <xx>` : same as `#include <xx.h>`

Standard header file structure

```
#ifndef HEADER_FLAG
#define HEADER_FLAG
    // Type declaration here...
#endif // HEADER_FLAG
```

Tips for header

1. One class declaration per header file
2. Associated with one source file in the same prefix of file name.
3. The contents of a header file is surrounded with `#ifndef` `#define` `#endif`

Initialization and Clean-Up

`Point::init()`

```
struct Point {
    void init(int x,int y);
    void print() const;
    void move(int dx,int dy);
    int x;
    int y;
};
Point a;    // a as a local variable
a.init(1,2);
a.move(2,2);
a.print();
```

- What is the initial value of `x` and `y` in `a`?
- What if the client forgot to call `init` before other functions?

Inital Value at Declaration

```
struct Point {
    void init(int x,int y);
    void print() const;
    void move(int dx,int dy);
    int x = 0;
    int y = 0;
};
Point a;
a.init(1,2);
a.move(2,2);
a.print();
```

- Value of other member variables defined before this member varible can be used here
- Return value of a function can be used here
    - What if a member varible is used in that function?

Guaranteed initialization with the constructor

- If a class has a constructor, the compiler automatically calls that constructor at the point an object is created, before client programmers can get their hands on the object.
- The name of the constructor is the same as the name of the class.

How a constructor does?

```
class X {
  int i;
public:
  X();
};
```

constructor

```
void f() {
  X a;
  // ...
}
```

a.X();

Constructors with arguments

- The constructor can have arguments to allow you to specify how an object is created, give it initialization values, and so on.

```
Tree(int i) {...}
Tree t(12);
```

- Constructor1.cpp

Initializer list

- Member variables can be initialized in place:

```
struct Point {
    int x=0;
    int y=0;
};
```

- Or at the initializer list of a constructor:

```
struct Point {
    Point(xx,yy);
    int x;
    int y;
};

Point::Point(int xx, int yy):x(xx),y(yy)
{}
```

The default constructor

- A default constructor is one that can be called with no arguments.

```
struct Y {
    float f;
    int i;
    Y(int a);   //  this is NOT a default ctor
};
Y y1[] = { Y(1), Y(2), Y(3) };  // OK
Y y2[2] = { Y(1) }; Y y3[7];    // ?
Y y4;                           // ?
```

“auto” default constructor

- If you have a constructor, the compiler ensures that construction always happens.
- If (and only if) there are no constructors for a class (struct or class), the compiler will automatically create one for you.
- Example: AutoDefaultConstructor.cpp

The Destructor

- In C++, cleanup is as important as initialization and is therefore guaranteed with the destructor.
- The destructor is named after the name of the class with a leading tilde (~).The destructor never has any arguments.

```
struct Y {
    ~Y();
};
```

When is a destructor called?

- The destructor is called automatically by the compiler when the object goes out of scope.
- The only evidence for a destructor call is the closing brace of the scope that surrounds the object.

Storage Allocation vs Initialization

- The compiler allocates all the storage for a scope at the opening brace of that scope.
- The constructor call doesn't happen until the sequence point where the object is defined.
- Examlpe: Nojump.cpp

Aggregate Initialization with Ctor

```
int a[5] = { 1, 2, 3, 4, 5 };
int b[6] = {5};
int c[] = { 1, 2, 3, 4 };
// sizeof c / sizeof *c
struct X { int i; float f; char c; };
X x1 = { 1, 2.2, 'c' };
X x2[3] = { {1, 1.1, 'a'}, {2, 2.2, 'b'} };
struct Y { float f; int i; Y(int a); };
Y y1[] = { Y(1), Y(2), Y(3) };
```

What we've learned today?

- How to decalare a class
  - Put functions into a struct
- The declaration and definition of a class
- The constructor and destructor