数据结构与算法

DATA STRUCTURE

—用面向对象方法与C++描述

信息管理与工程学院

2017 - 2018 第一学期

课程信息

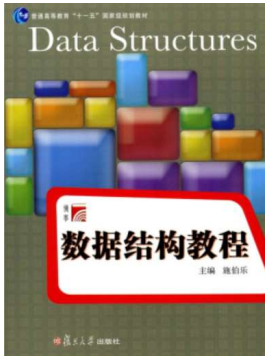
- 助教：TBD
- 评分标准：
  - 期末闭卷笔试（约60%左右）
  - 项目实践(功能实现，易读性，性能测试)
  - 作业及出勤情况
- 教师：胡浩栋
- 答疑时间：星期二下午3：30 - 5：30
- 答疑地点：信管学院606

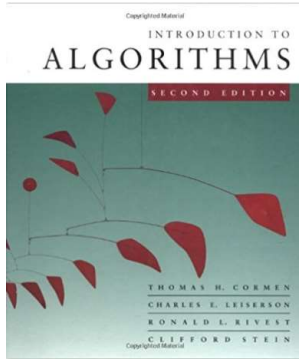
参考教材

- 数据结构教程

  复旦大学出版社

  施伯乐





- Introduction to Algorithms,
  3rd Edition,
  MIT Press

为什么上这门课

- 算法是计算机科学的基础
- 算法很有用，培养解决问题的方法和思考
- 找工作面试都问算法
- 注重理论和应用结合

- 前提：上过程序设计基础，懂C++，或者热爱编程

课程内容

- 复杂度分析
- 数组(Array)
- 链表(LinkList)
- 字符串(String)
- 栈(Stack)
- 队列(Queue)
- 递归(Recursion)
- 树(Tree)
- 查找和索引(Searching, Hashing)
- 图(Graph)
- 排序(Heap)
- 外部排序

课程目的

- 理解，实现，正确使用在各种数据结构上的操作
- 分析代码的复杂度，来评估实现的优劣
- 只有自己制造过的轮子，才是自己的轮子

基本概念：

- **数据** (Data)：
  客观事物抽象到计算机，能被计算机识别的集合

- **数据元素** (Data Element)
  数据的基本单位，一般当不可分割部分来操作

- **数据操作**
  对数据元素之间按逻辑关系进行运算的一次操作

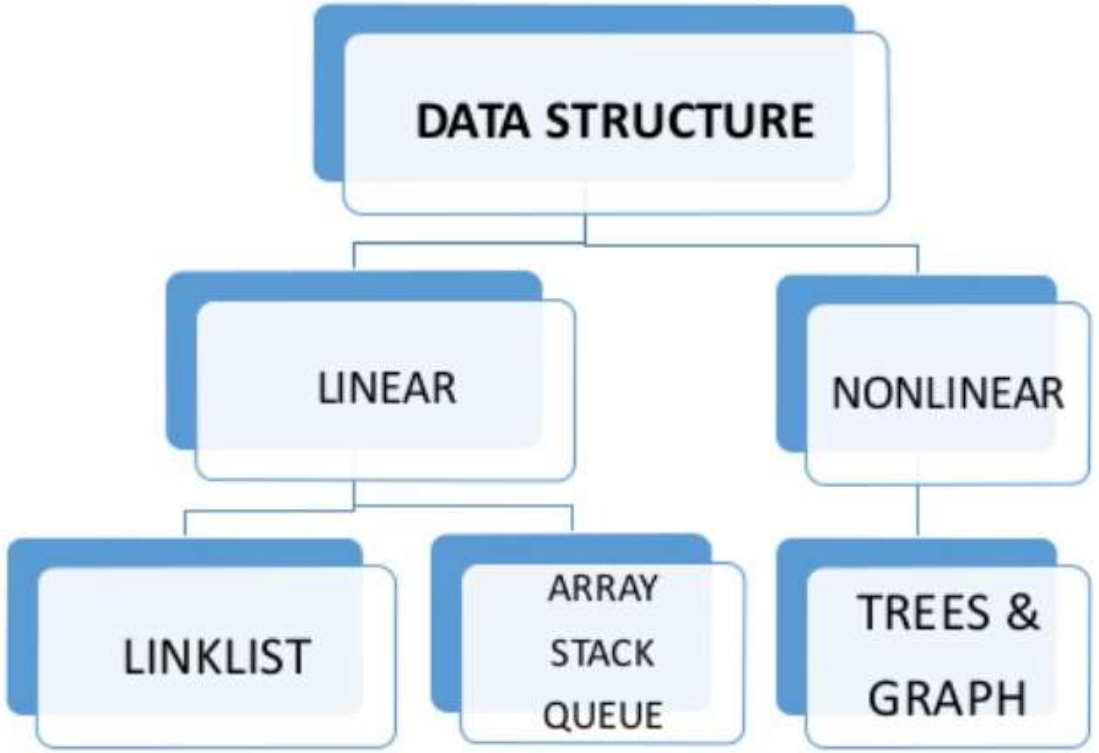
什么是数据结构

能有效地存储和操作在计算机里的**数据元素**

- 既是一种对数据元素存储方式的管理
- 也是一种方法，能对存储的数据元素进行有效的**操作**

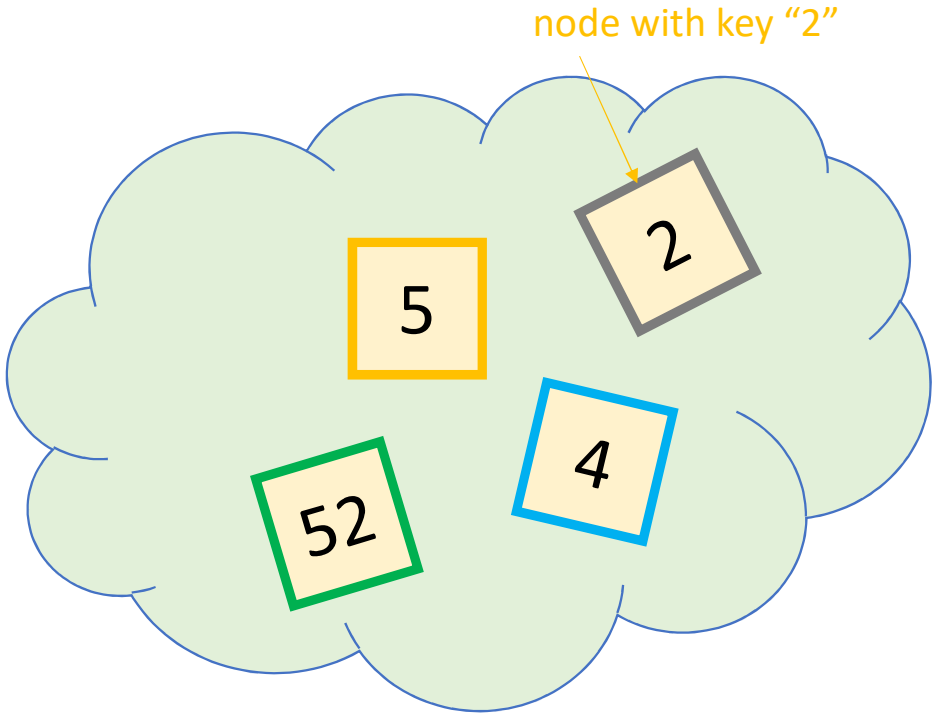
数据结构例子

- 基本的数据结构
  - 数组(Array)
  - 链表(LinkList)
- 衍生的数据结构
  - 栈(Stack)
  - 队列(Queue)
  - 树(Tree)
  - 图(Graph)
  - 堆(Heap)
  - etc



一种问题原型

比如需要维护亿万的整数数据（身份证），使得能快速查找，快速删除和快速插入



“有效”的标准

解法所需的资源利用情况

- 时间复杂度
- 空间复杂度

复杂度描述：Big-O术语

- f(n) = O(g(n)) 指的是函数f(n)增长的上界
- 例子

  $n + 12 = O(n)$

  $n^2 + 3n – 2 = O(n^2)$

  $n^3 + 10n^2\log(n) - 15n = O(n^3)$

  $2^n + n^2 = O(2^n)$

正式定义

- Let f, g: N -> N,

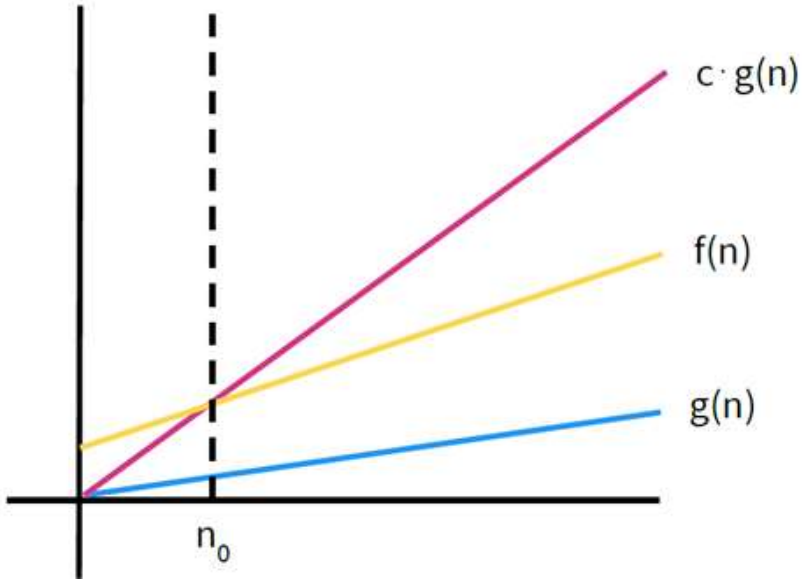
  Then f(n) = O(g(n)) if and only if

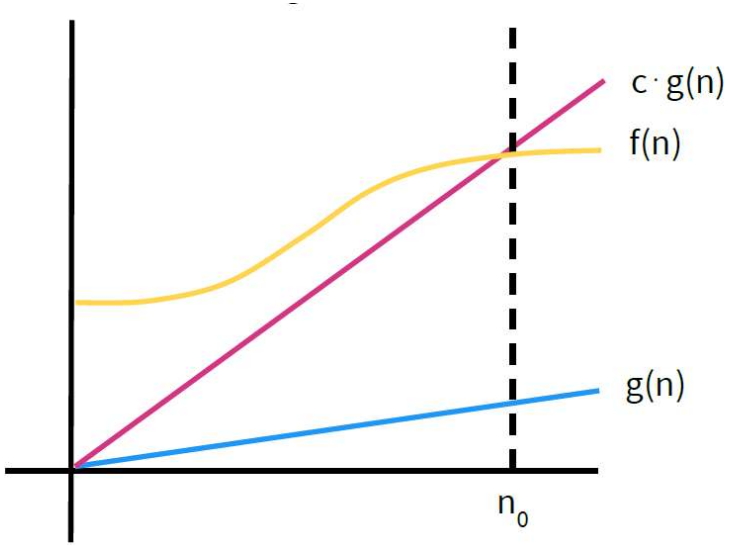
$$\exists n_0 \in N, c \in R, \quad \forall n \in N, \quad n \geq n_0 \rightarrow f(n) \leq c \cdot g(n)$$

也就是说f(n)最多是和g(n)一样大，除了一个常数倍数c。

Big- O术语

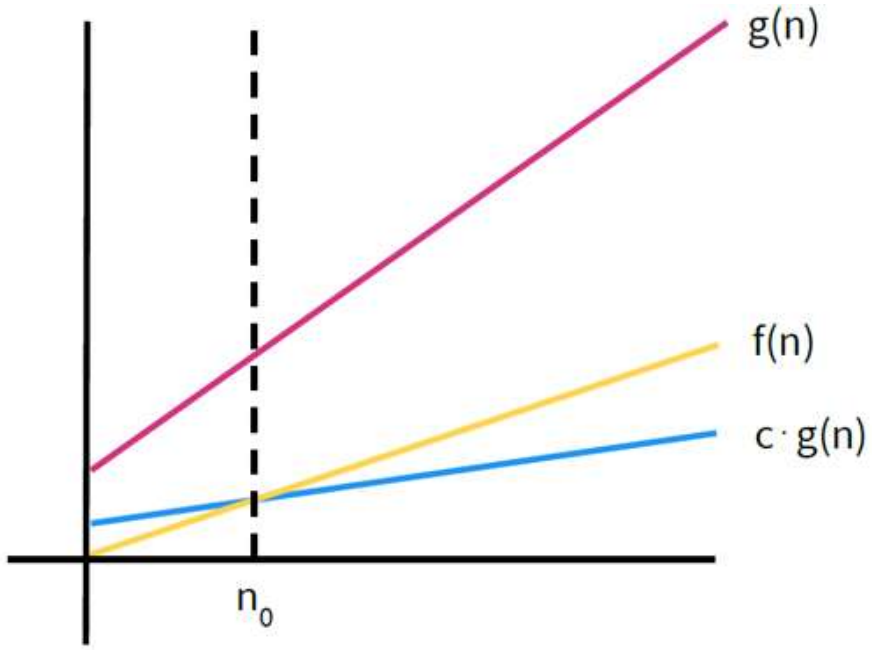
$$f(n) = O(g(n)) \quad \text{iff} \quad \exists n_0 \in N, c \in R, \quad \forall n \in N, \quad n \geq n_0 \rightarrow f(n) \leq c \cdot g(n)$$





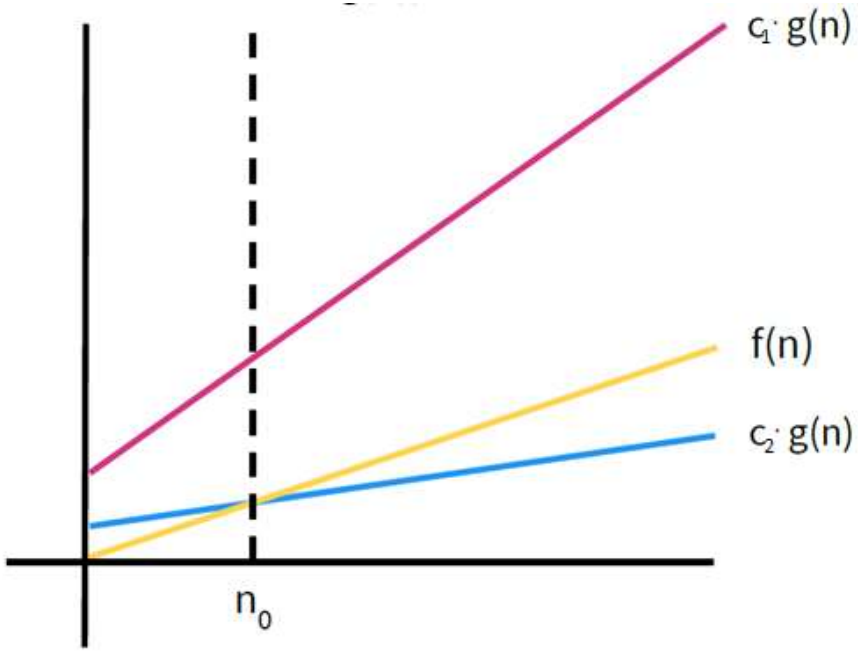
Big- Ω术语

$f(n) = \Omega(g(n))$ iff $\exists n_0 \in N, c \in R, \forall n \in N, n \geq n_0 \rightarrow f(n) \geq c \cdot g(n)$



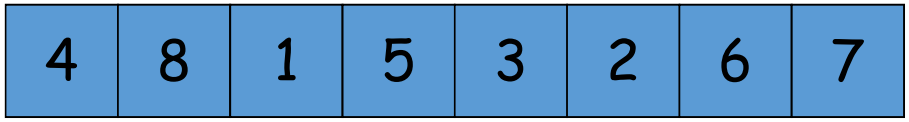
Big- Θ术语

$f(n) = \Theta(g(n))$ iff $\exists n_0 \in N, c \in R, \forall n \in N, n \geq n_0 \rightarrow f(n) \leq c_1 \cdot g(n)$ and $f(n) \geq c_2 \cdot g(n)$



复杂度例子：排序问题

- 给定一个无序元素数组Array $\{a_1, a_2, \ldots, a_n\}$，要求比较排序成$\{a_{i1}, a_{i2}, \ldots, a_{in}\}$，使得 $a_{i1} < a_{i2} < \ldots < a_{in}$。



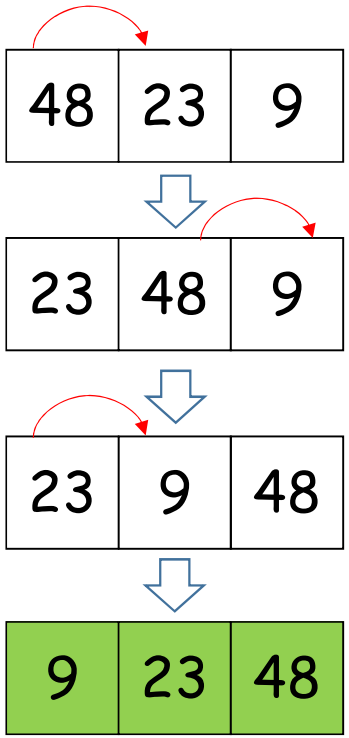
基于比较(Comparison)的排序

可比较的数学定义：

- ∀a, b, a > b or a = b or a < b
- If a > b, b > c, then a > c

冒泡排序 BubbleSort

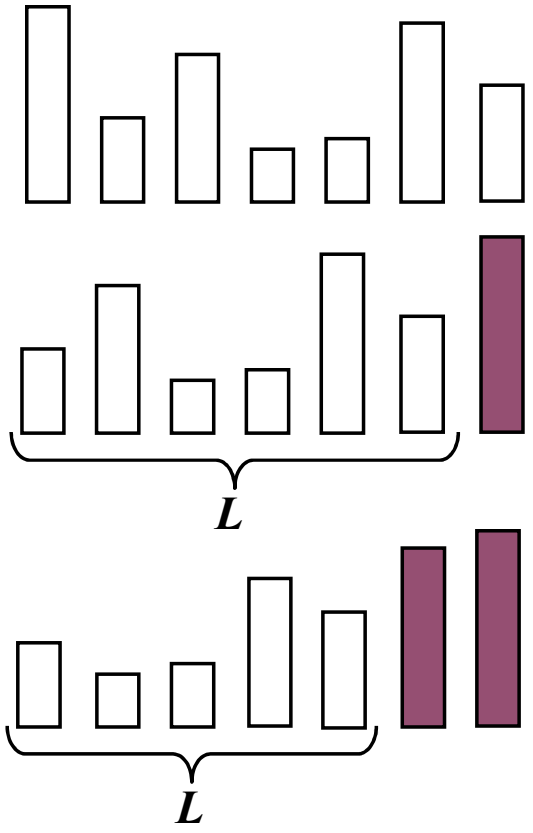
- Naive：比较相邻的元素，如果第一个比第二个大，就交换他们两个。
- Idea：通过两两比较交换，像水中的泡泡一样，大的先冒出来，小的后冒出来。



BubbleSort复杂度

$$
\begin{aligned}
T(n) &= T(n-1) + (n-1) \\
&= T(n-2) + (n-2) + (n-1) \\
&= \ldots \\
&= 1 + \ldots + (n-2) + (n-1) \\
&= O(n^2)
\end{aligned}
$$

- 复杂度高，实际中没人用
- 名字取得好
- 适合入门算法

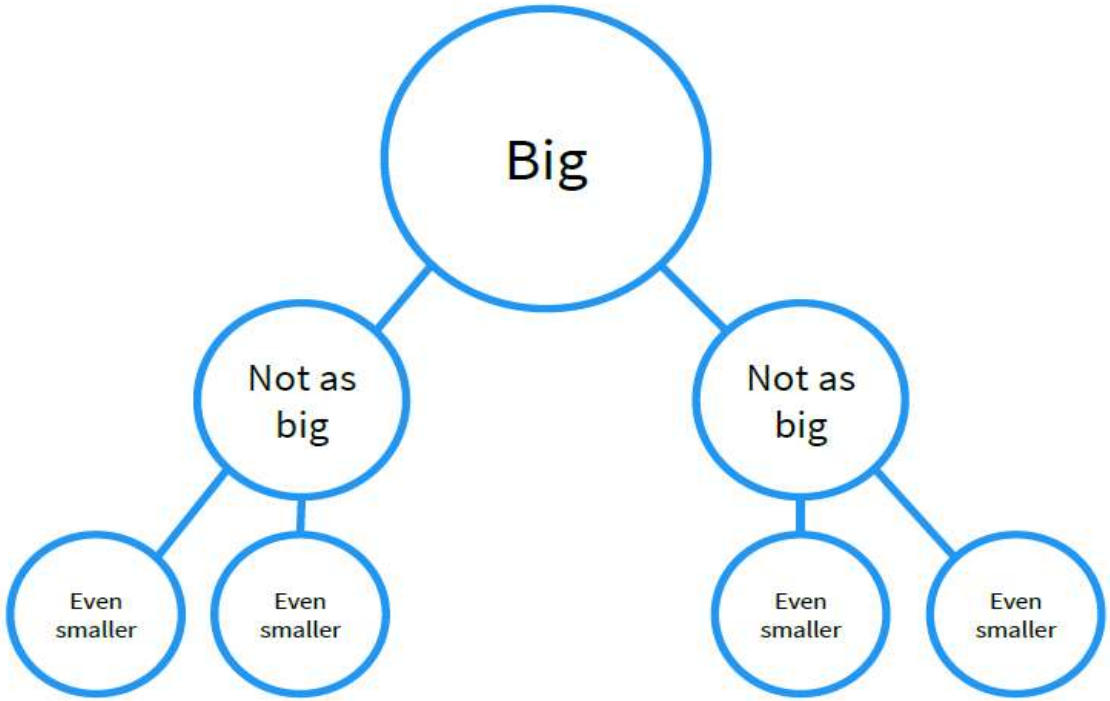


如何改进？

Idea: 分治法 Divide & Conquer

- 基本思想是：
  - 将原问题分解为若干个规模更小但结构与原问题相似的子问题。
  - 然后递归地解这些子问题。
  - 最后将这些子问题的解组合为原问题的解。
- 计算机基础算法三大思想之一。

分治法 – 从下往上的方法

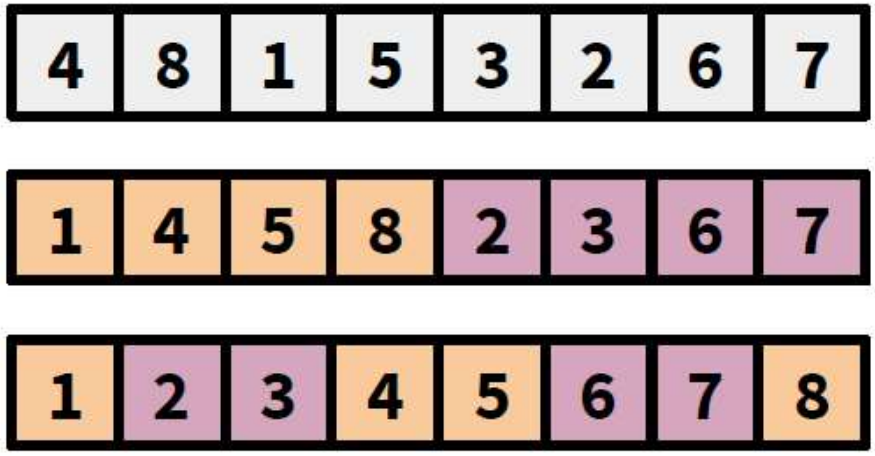


改进的方法：**MergeSort**采用了一种分治的策略。

归并排序 （MergeSort）

对长度为8的数组归并排序

- 分别对A[0,3]和A[4,7]排序
- 然后把两个排好序的半段归并



归并算法

```
algorithm mergesort(list A):
  if length(A) ≤ 1:
    return A
  let left = first half of A
  let right = second half of A
  return merge(
    mergesort(left),
    mergesort(right)
  )
```

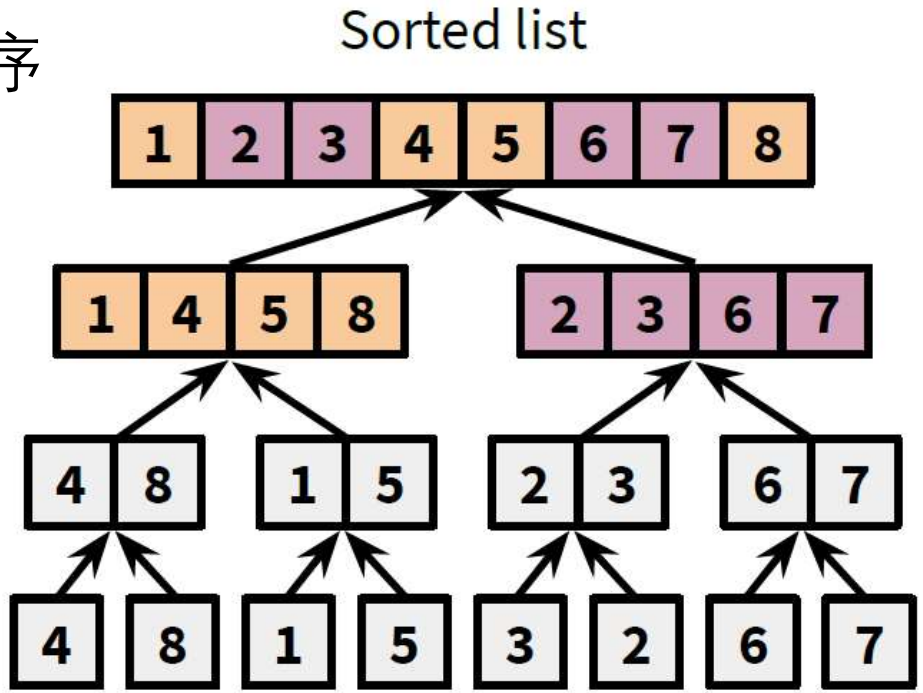
归并算法（cont）

```
algorithm merge(list A, list B):
  let result = []
  while both A and B are nonempty:
    if head(A) < head(B):
      append head(A) to result
      pop head(A) from A
    else:
      append head(B) to result
      pop head(B) from B
  append remaining elements in A to result
  append remaining elements in B to result
  return result
```

Total work: O(a+b), where a and b are the lengths of lists A and B.

Mergesort复杂度分析

- T(n)是用对长度为n的数组归并排序
  - 分别对A[0,3]和A[4,7]排序
  - 复杂度2T(n/2)
  - 然后把两个排好序的半段归并
  - 复杂度Θ(n)
  - T(0) = Θ(1)
  - T(n) = 2T(n/2) + Θ(n)

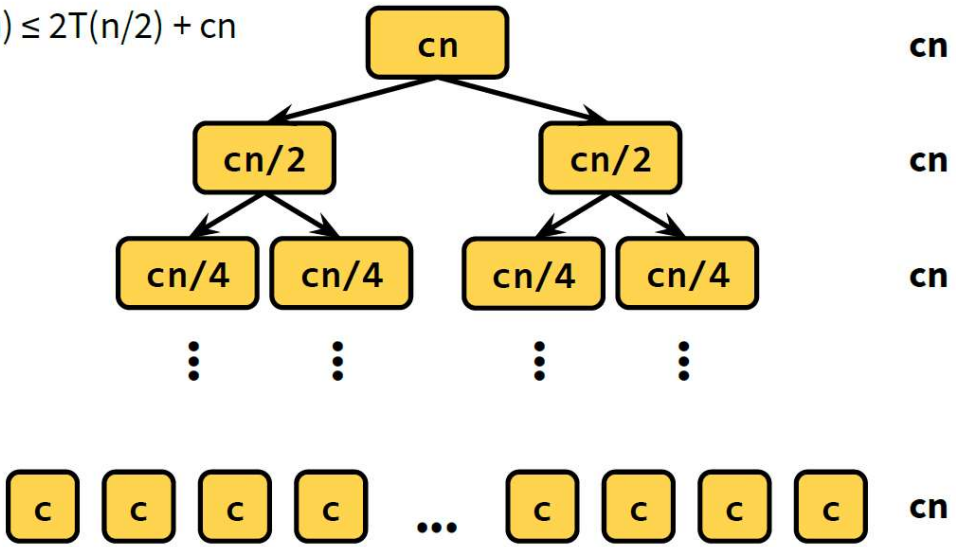


递归求解

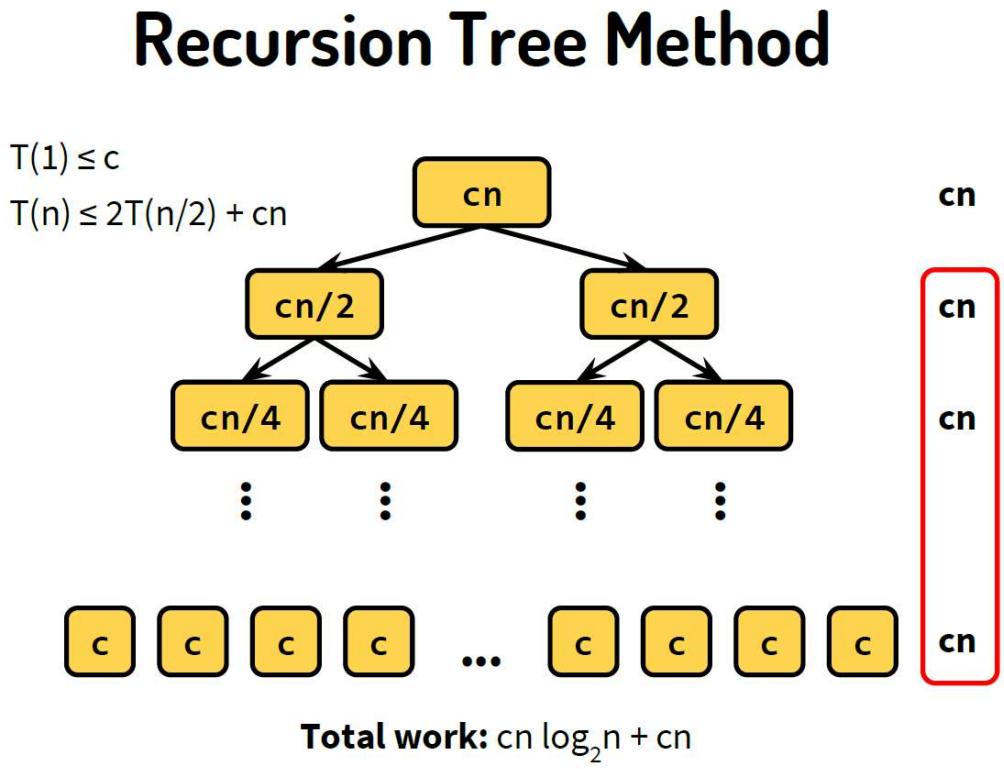
Recursion Tree Method

$T(1) \le c$

$T(n) \le 2T(n/2) + cn$

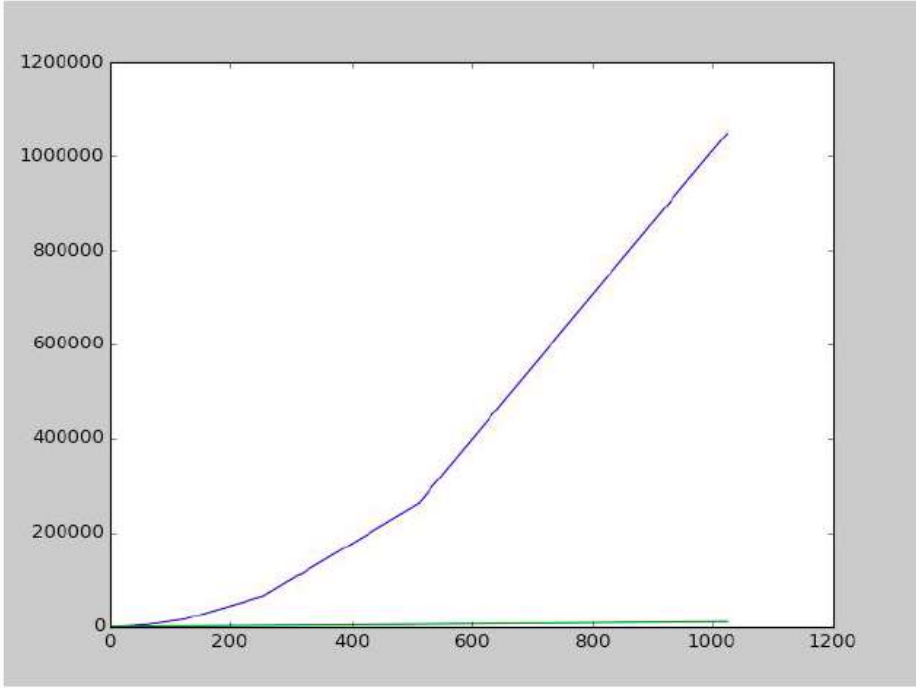


递归求解


Recursion Tree Method
T(1) ≤ c
T(n) ≤ 2T(n/2) + cn
cn
cn/2
cn/2
cn/4
cn/4
cn/4
cn/4
c
c
c
c
...
c
c
c
c
cn
cn
cn
cn
Total work: cn log2n + cn

冒泡排序 vs 归并排序

- $O(n^2)$ vs O(nlogn)



Master Theorem

- if $T(n) = aT(n/b) + f(n)$ then

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b a - \varepsilon}\right) \\ \Theta\left(n^{\log_b a} \log n\right) & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right) & f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) \text{ AND } af(n/b) < cf(n) \text{ for large } n \end{cases} \quad \begin{matrix} \varepsilon > 0 \\ c < 1 \end{matrix}$$

- $T(n) = 2T(n/2) + \Theta(n)$

  a = 2
  b = 2
  d = 1

  $T(n) = O(n^d \log n) = O(n \log n)$

回顾C++重点

类（class）

- 构造函数(constructor), 析构函数(destructor)

Member function	Typical form for class C:
Default constructor	C::C();
Destructor	C::~C();
Copy constructor	C::C (const C&);
Copy assignment	C& operator= (const C&);
Move constructor （C++11）	C::C (C&&);
Move assignment （C++11）	C& operator= (C&&);

类（class）

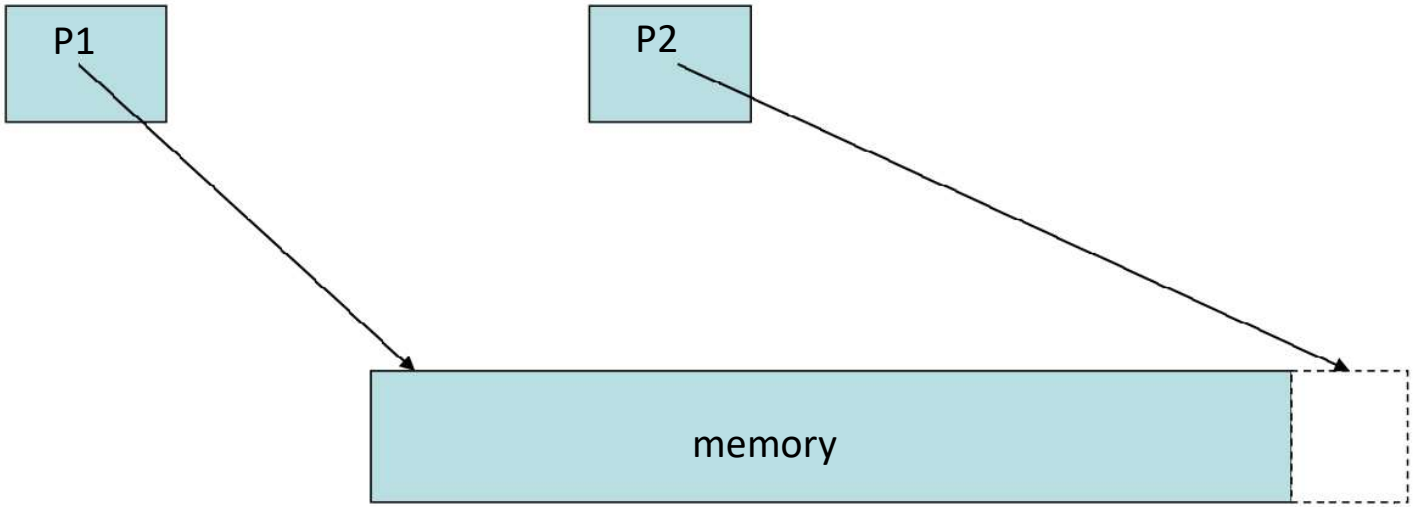
Member function	implicitly defined:	default definition:
Default constructor	if no other constructors	does nothing
Destructor	if no destructor	does nothing
Copy constructor	if no move constructor and no move assignment	copies all members
Copy assignment	if no move constructor and no move assignment	copies all members
Move constructor	if no destructor, no copy constructor and no copy nor move assignment	moves all members
Move assignment	if no destructor, no copy constructor and no copy nor move assignment	moves all members

Rule of three：

If a class defines a copy constructor, a copy assignment operator, or a destructor, then it should define all three.

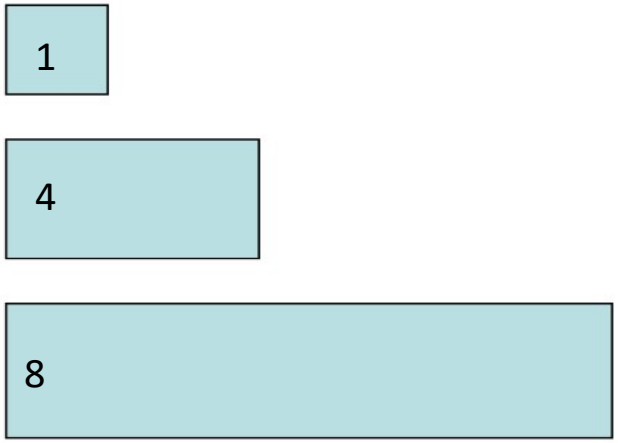
内存模型

- 指针



内存模型（内置类型）

- char
- short
- int
- long
- (long long)
- float
- double
- long double
- T* (pointer)
- T& (implemented as pointer)



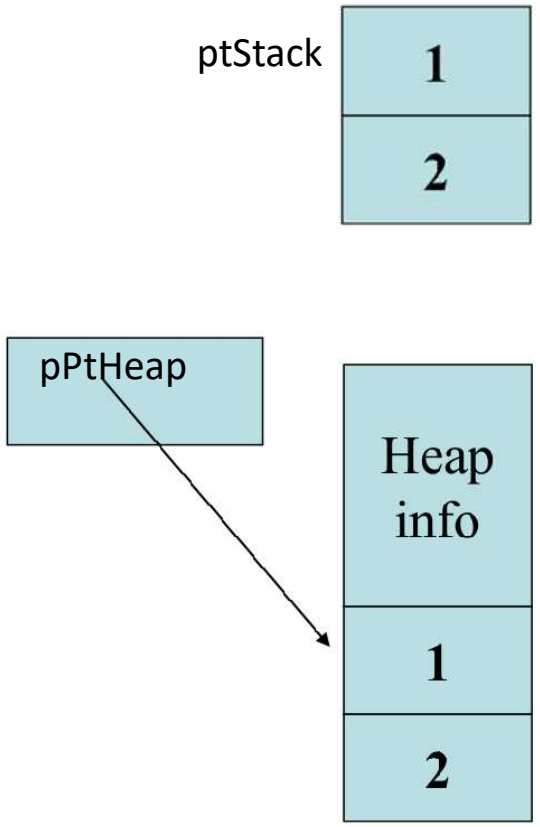
内存模型（正常Class）

- class **Point**

```
 {
        int x, y;
       // int getX();
  };
```

Point ptStack(1,2);

Point* pPtHeap = new **Point(1,2)**;



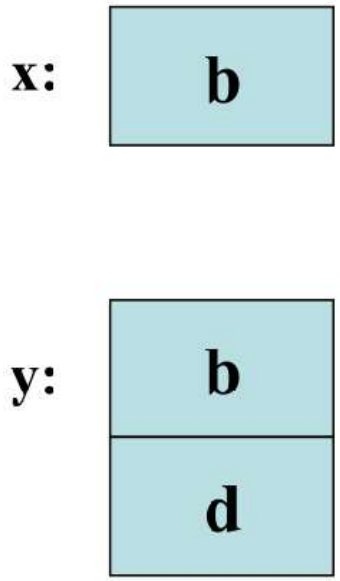
内存模型（Derived Class）

- class **Base**

```
{
    int b;
};
```

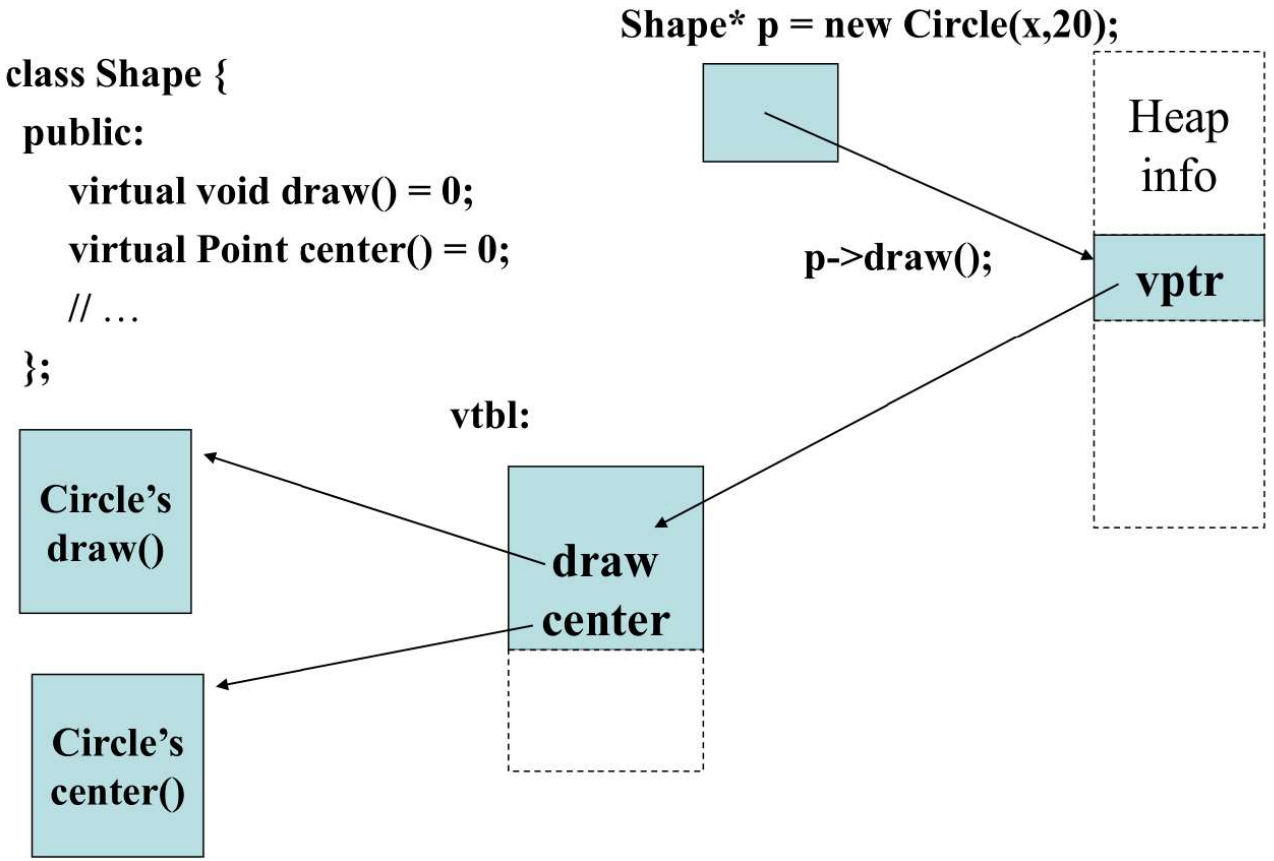
- class **Derived** : public Base

```
{
    int d:
};
```



Base x;
Derived y;

内存模型（多态polymorphic）



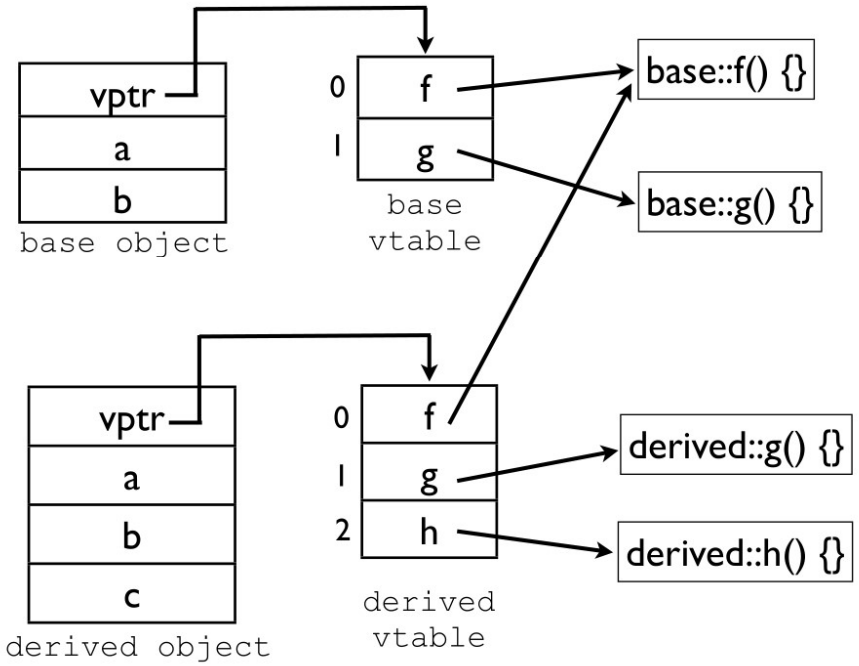
The vtable

```
struct base
{
    virtual void f();
    virtual void g();
    int a,b;
};

struct derived : base
{
    virtual void g();
    virtual void h();
    int c;
};

void poly(base * ptr)
{
    ptr->f();
    ptr->g();
}

int main()
{
    poly(&base());
    poly(&derived());
}
```



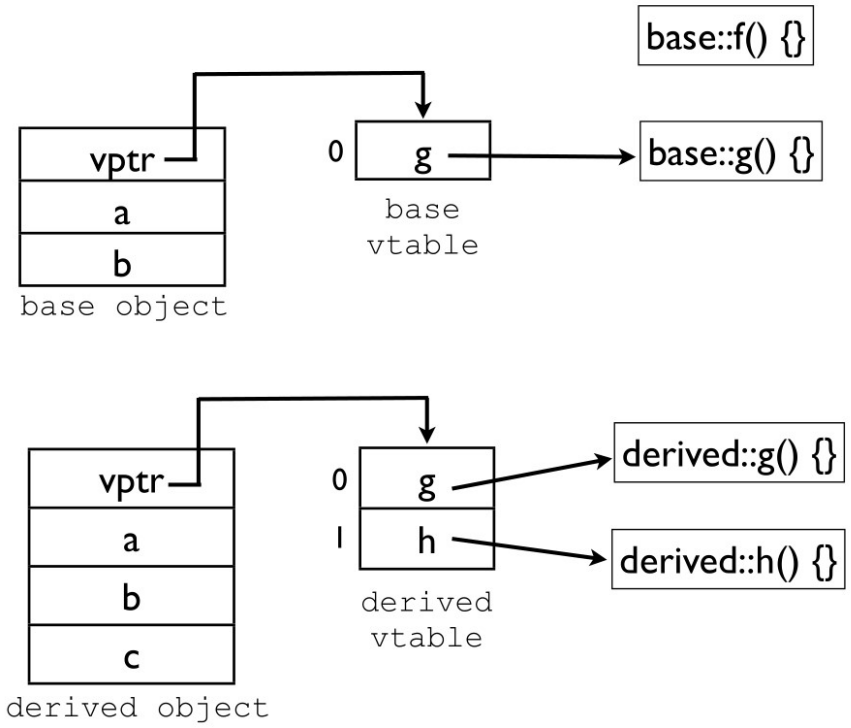
The vtable

```
struct base
{
            void f();
    virtual void g();
    int a,b;
};

struct derived : base
{
    virtual void g();
    virtual void h();
    int c;
};

void poly(base * ptr)
{
    ptr->f();
    ptr->g();
}

int main()
{
    poly(&base());
    poly(&derived());
}
```



模板(template)

- 目的：实现软件重用

- 定义

适合多种数据类型的类定义或算法，在特定环境下通过简单地代换，变成针对具体某种数据类型的类定义或算法。

格式

- 定义template

template <class identifier> function_declaration;
template <typename identifier> function_declaration;

```
template <class T>
T GetMax (T a, T b) {
  T result;
  result = (a>b)? a : b;
  return (result);
}
```

```
template <class T, class U>
T GetMin (T a, U b) {
  return (a<b?a:b);
}
```

- 使用template

function_name <datatype> (parameters);

```
int x,y;
GetMax <int> (x,y);
```

```
int i,j;
long l;
i = GetMin<int,long> (j,l);
```

Class template

```
1  // class templates
2  #include <iostream>
3  using namespace std;
4
5  template <class T>
6  class mypair {
7      T a, b;
8    public:
9      mypair (T first, T second)
10       {a=first; b=second;}
11     T getmax ();
12 };
13
14 template <class T>
15 T mypair<T>::getmax ()
16 {
17   T retval;
18   retval = a>b? a : b;
19   return retval;
20 }
```

```
mypair<int> myobject (115, 36);
```

```
mypair<double> myfloats (3.0, 2.18);
```

```
1 template <class T>
2 T mypair<T>::getmax ()
```

Q&A

Thanks!