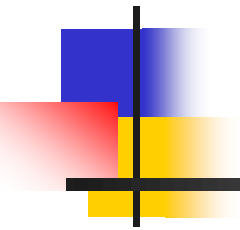
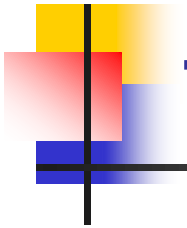


Data Structures and Algorithm Analysis

Department of Computer Science



The Course

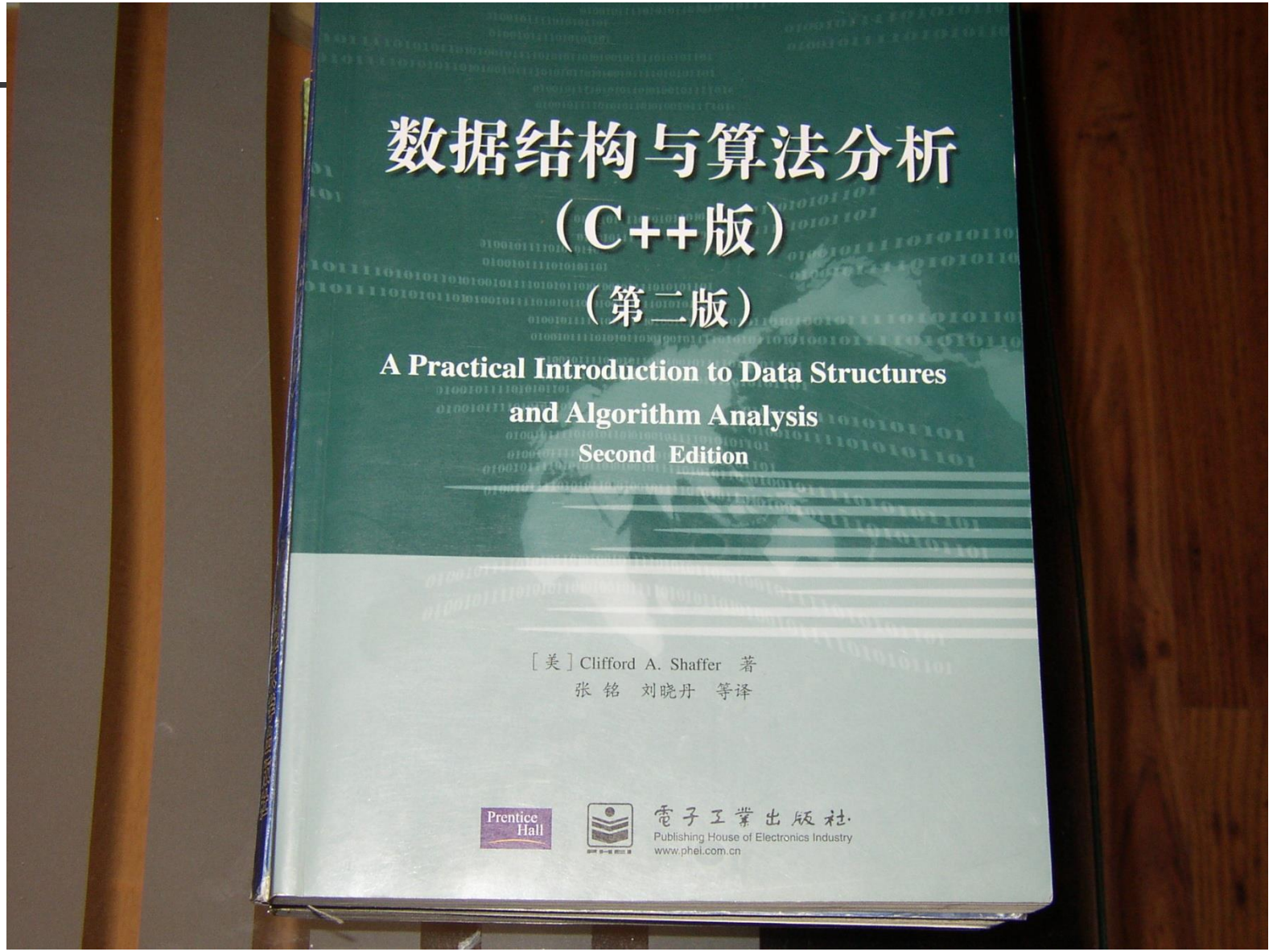
- **Purpose: a rigorous introduction to the design and analysis of data structures and algorithms**
  - **Not a lab or programming course**
  - **Not a math course, either**
- **Prerequisites:**
  - **C or C++**
  - **Maths**

The Course

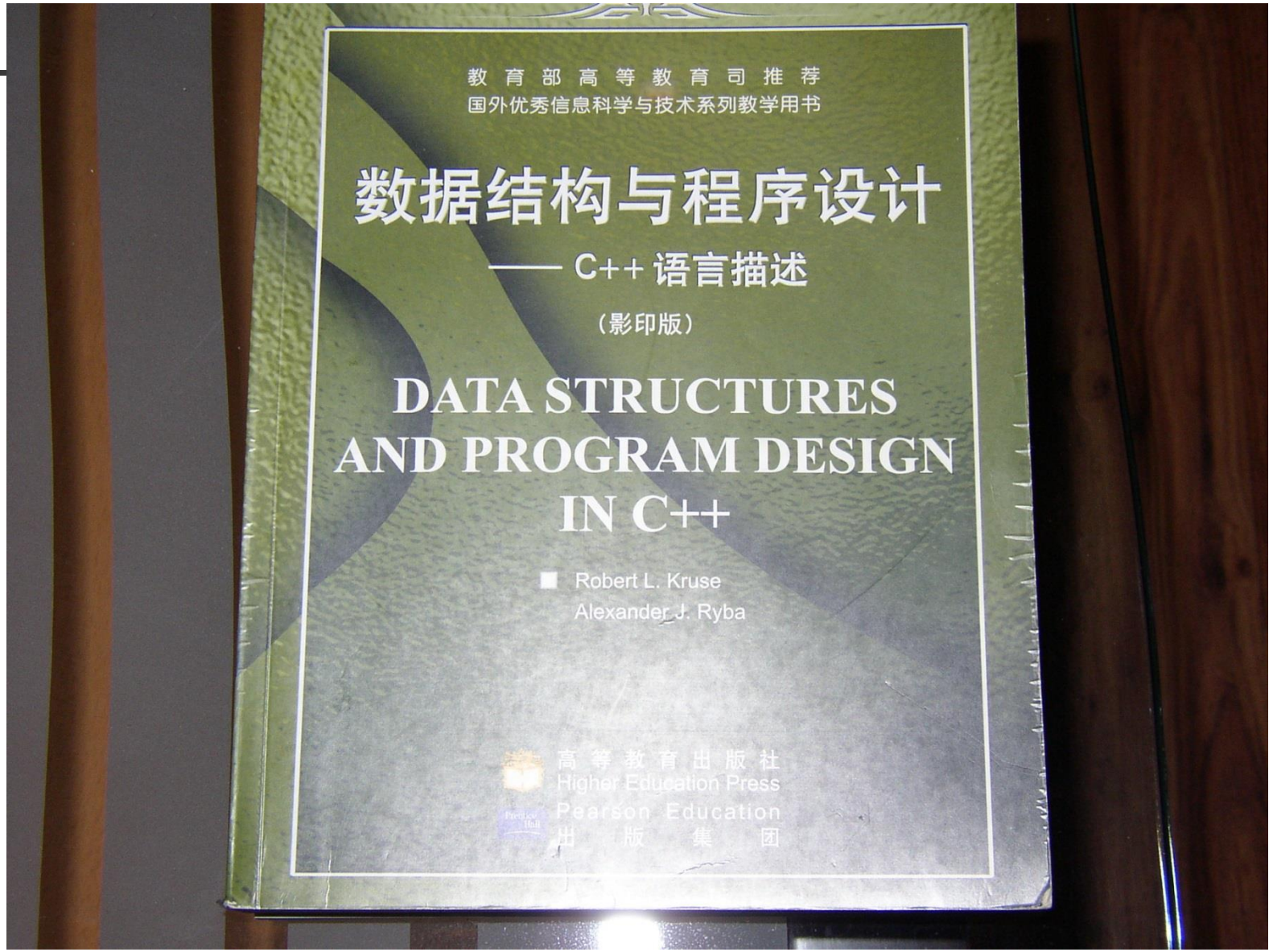
- **Grading policy:**
  - **Final: 70~80%**
  - **Exercises: 20~30%**
- **Format**
  - **Three hours lectures/week**
  - **Six or seven exercises**
  - **final exam**

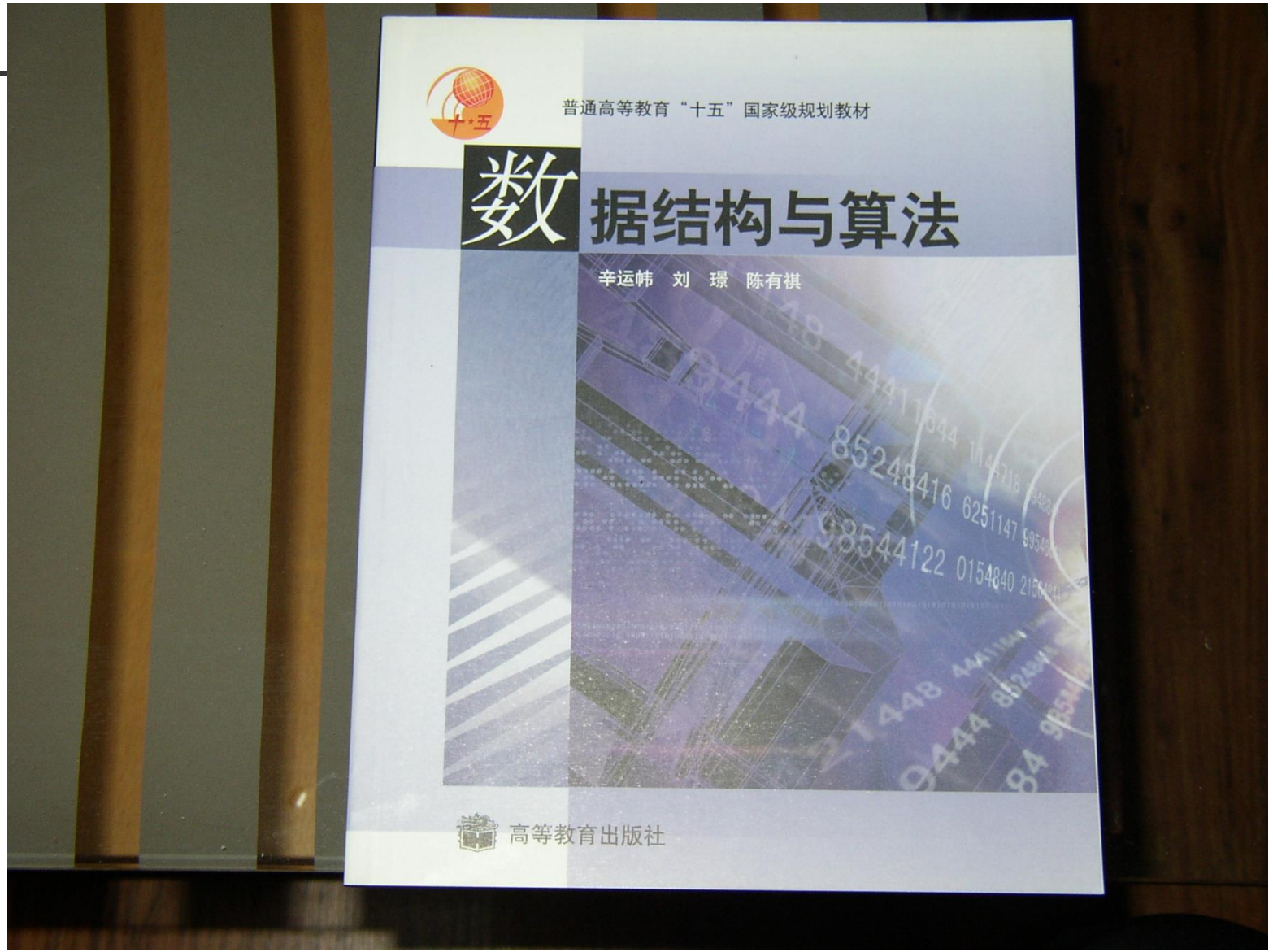
References

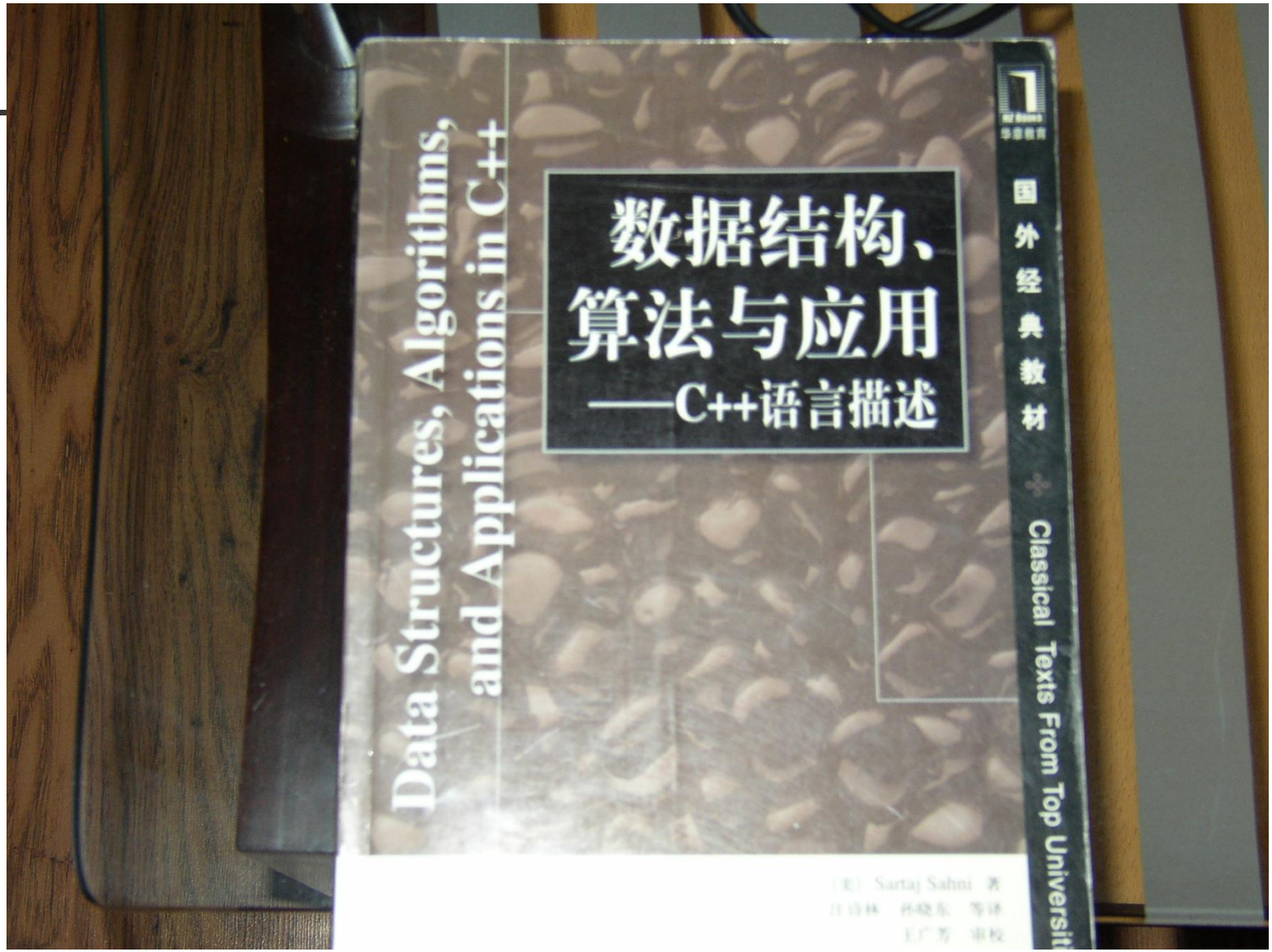
- 数据结构与程序设计
  - *Robert L.Kruse & Alexander J.Ryba*
  - *高等教育出版社*
- 数据结构与算法分析
  - *DATA STRUCTURES AND ALGORITHM ANALYSIS*
  - *CLIFFORD A. SHAFFER著　　张铭 刘晓丹译*
  - *电子工业出版社　PRENTICE HALL出版公司*
- 数据结构（C语言版）
  - *严蔚敏 吴伟民 编著*
  - *清华大学出版社*


数据结构与算法分析
（C++版）
（第二版）
A Practical Introduction to Data Structures
and Algorithm Analysis
Second Edition
[美] Clifford A. Shaffer 著
张 铭 刘晓丹 等译
Prentice Hall
電子工業出版社
Publishing House of Electronics Industry
www.phei.com.cn


（C语言版）
数据结构
严蔚敏 吴伟民 编著
清华大学出版社


教育部高等教育司推荐
国外优秀信息科学与技术系列教学用书
数据结构与程序设计
——C++语言描述
（影印版）
DATA STRUCTURES
AND PROGRAM DESIGN
IN C++
Robert L. Kruse
Alexander J. Ryba
高等教育出版社
Higher Education Press
Pearson Education
出版集团


普通高等教育“十五”国家级规划教材
数据结构与算法
辛运帏 刘 璟 陈有祺
高等教育出版社


数据结构、
算法与应用
——C++语言描述
Data Structures, Algorithms,
and Applications in C++
国外经典教材
Classical Texts From Top Universit

General Programming Standards

- **Internal Documentation Requirements:**
  - **Each source and header file must begin with a comment block**
  - **The source file containing main() must include a comment block**
  - **Each function must be accompanied by a header comment**

General Programming Standards

- **Internal Documentation Requirements:**
  - **Declarations of all local variables and constants must be accompanied by a brief description of purpose.**
  - **Major control structures should be preceded by a block comment**
  - **Use a sensible, consistent pattern of indentation and other formatting style**
  - **Each class must have a header comment block**

General Programming Standards

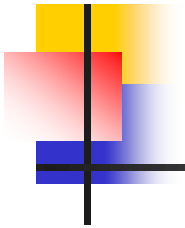
- **Procedural Coding Requirements:**
  - **Identifier names should be descriptive.**
  - **When a constant is appropriate, use a named constant instead of a "magic number".**
  - **Use enumerated types for internal labeling and classification of state.**
  - **Do not use global or file-scoped variables under any circumstances. It IS permissible to use globally scoped type definitions (including class declarations).**

General Programming Standards

- **Procedural Coding Requirements:**
  - **Pass function parameters with appropriate access.**
    - **Use pass-by-reference or pass-by-pointer only when the called function needs to modify the value of the actual parameter.**
    - **Use pass-by-constant-reference or pass-by-constant-pointer when passing large structures that are not modified by the called function.**
    - **Use pass-by-value when the called function must modify the formal parameter (internal to the call) but the actual parameter should remain unmodified.**

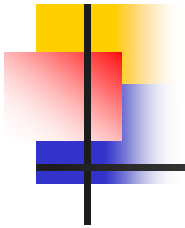
General Programming Standards

- **Procedural Coding Requirements:**
  - **Store character data (aside from single characters) in string objects, rather than char arrays.**
  - **Use new-style C++ at all times; Do not mix old and new style C++ headers.**
  - **Use stream I/O instead of C-style I/O.**



General Programming Standards

- **Object-Oriented Coding Requirements:**
  - **Use classes where they are appropriate. Do not implement struct types with member functions.**
  - **When specified, use a template class for container structures such as lists, trees, etc.**
  - **Design each class to have a coherent set of responsibilities.**
  - **Except for node classes used only with an encapsulating class, all data members of a class should be private.**



General Programming Standards

- **Object-Oriented Coding Requirements:**
  - **In many cases, some of the member functions of a class should also be private. Watch for that situation.**
  - **If a class data member is a pointer to dynamically allocated memory, implement a destructor to deallocate that memory.**
  - **If a class data member is a pointer to dynamically allocated memory, implement a deep copy constructor and assignment operator overload.**
  - **Use inheritance only when it makes sense to do so.**

匈牙利命名法

- **规则1**　　标识符的名字以一个或者多个小写字母开头，用这些字母来指定数据类型。下面列出了常用的数据类型的标准前缀：

前缀数据类型

c　字符（char）
s　整数（short）
cb　用于定义对象（一般为一个结构）尺寸的整数
n　整数（integer）
sz　以’\0’结尾的字符串
b　字节
i　int（整数）

匈牙利命名法

- x　短整数（坐标x）
  y　短整数（坐标y）
  f　BOOL
  w　字（WORD，无符号短整数）
  l　长整数（long）
  h　HANDLE（无符号int）
  m　类成员变量
  fn　函数（function）
  dw 双字（DWORD，无符号长整数）

匈牙利命名法

- **规则2**　在标识符内，前缀以后就是一个或者多个首字母大写的单词，这些单词清楚地指出了源代码内那个对象的用途。

Java中的命名约定

- 类及接口名
  - 类是Java中最重要的数据结构，类名一般为多个单词并在一起，中间不加分隔符，且首字符大写。为方便阅读，一般地类名中每个字的首字母也大写。
  - 比如：HelloWorld、Customer、MergeSort等。
  - 接口是一种特殊的类，它的命名约定与类名相同。

Java中的命名约定

- 方法名及变量名
  - 这两种标识符都是程序中的重要元素，常常是多个单词并在一起，且首字符小写，其他各字的首字大写。
  - 因方法是实现功能的核心部分，所以方法名中常常含有描述主要操作的动词，及相关的操作对象。虽然在语法上并没有严格规定，但尽量不要在方法名和变量名中使用下划线，还要避免使用美元符号（$），因为该字符对内层类有特殊含义。
  - 例如如下的方法名都是常用的：**getName**、**setAddress**、**searchObject**、**raiseSalary**等。
  - 可以定义的变量名如：**balance**、**orders**、**byPercent**等。

Java中的命名约定

- 常量名
  - 简单类型常量的名字应该全部为大写字母，字与字之间用下划线分隔，对象常量可使用混合大小写。
  - 例如可以定义这样的常量名：**PI**、**BLUE_COLOR**、**MAX_VALUE**等。

The Need for Data Structures

- **Data structures organize data**
  =>more efficient programs.
  - **More powerful computers**
    =>more complex applications.
  - **More complex applications demand more calculations.**
  - **Complex computing tasks are unlike our everyday experience.**

The Need for Data Structures

- **Any organization for a collection of records can be searched, processed in any order, or modified.**
  - **The choice of data structure and algorithm can make the difference between a program running in a few seconds or many days.**

Efficiency

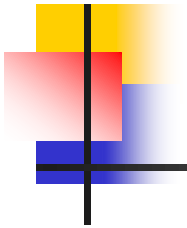
- **A solution is said to be efficient if it solves the problem within its resource constraints.**
  - **Space**
  - **time**
- **The cost of a solution is the amount of resources that the solution consumes.**

Selecting a Data Structure

- **Select a data structure as follows:**
  - **1. Analyze the problem to determine the resource constraints a solution must meet.**
  - **2. Determine the basic operations that must be supported. Quantify the resource constraints for each operation.**
  - **3. Select the data structure that best meets these requirements.**

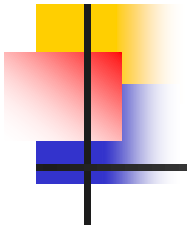
Some questions to ask:

- Are all data inserted into the data structure at the beginning, or are insertions interspersed with other operations?
- Can data be deleted?
- Are all data processed in some well-defined order, or is random access allowed?



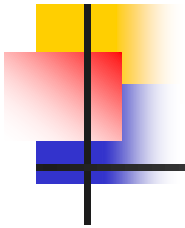
Data Structure Philosophy

- **Each data structure has costs and benefits.**
- **Rarely is one data structure better than another in all situations.**
- **A data structure requires:**
  - **space for each data item it stores,**
  - **time to perform each basic operation,**
  - **programming effort.**



Data Structure Philosophy

- **Each problem has constraints on available space and time.**
- **Only after a careful analysis of problem characteristics can we know the best data structure for the task.**



Goals of this Course

- **Reinforce the concept that there are costs and benefits for every data structure.**
- **Learn the commonly used data structures. These form a programmer's basic data structure :toolkit.**
- **Understand how to measure the effectiveness of a data structure or program.**
  - **These techniques also allow you to judge the merits of new data structures that you or others might invent.**

Definitions

- A ***type*** is a set of values.
- A ***data type*** is a type and a collection of operations that manipulate the type.
- A ***data item*** or ***element*** is a piece of information or a record.
- A data item is said to be a ***member*** of a data type.
- A ***simple data item*** contains no subparts.
- An ***aggregate data item*** may contain several pieces of information.

Abstract Data Types

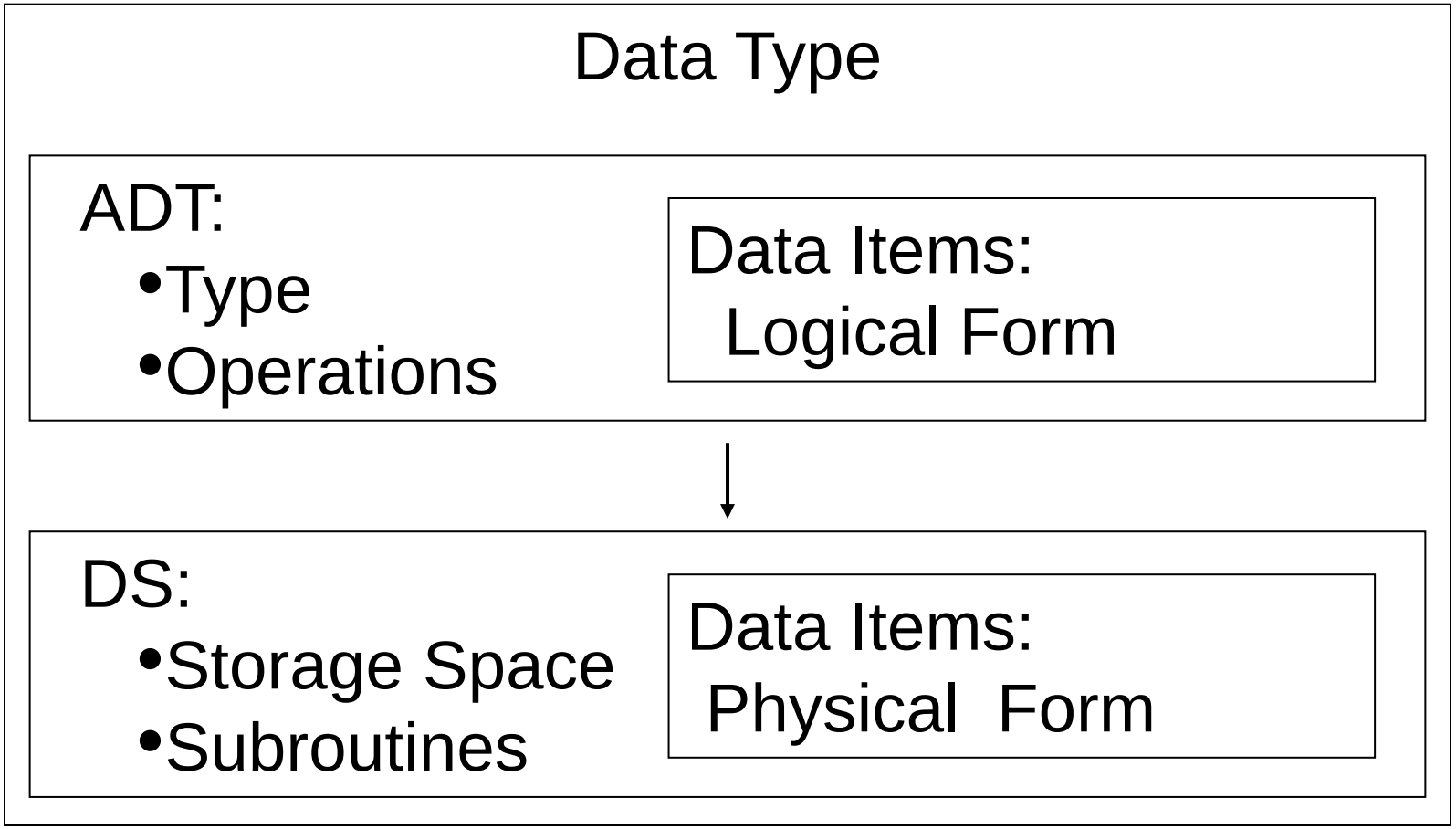
- ***Abstract Data Type (ADT):*** **a definition for a data type solely in terms of a set of values and a set of operations on that data type.**
- **Each ADT *operation* is defined by its inputs and outputs.**
- ***Encapsulation*: hide implementation details**

Abstract Data Types

- A ***data structure*** is the physical implementation of an ADT.
  - Each operation associated with the ADT is implemented by one or more subroutines in the implementation.
- Data structure usually refers to an organization for data in main memory.
- ***File structure***: an organization for data on peripheral storage, such as a disk drive or tape.
- An ADT manages complexity through abstraction: metaphor.

Logical vs. Physical Form

- **Data items have both a logical and a physical form.**
- ***Logical form*: definition of the data item within an ADT.**
- ***Physical form*: implementation of the data item within a data structure.**

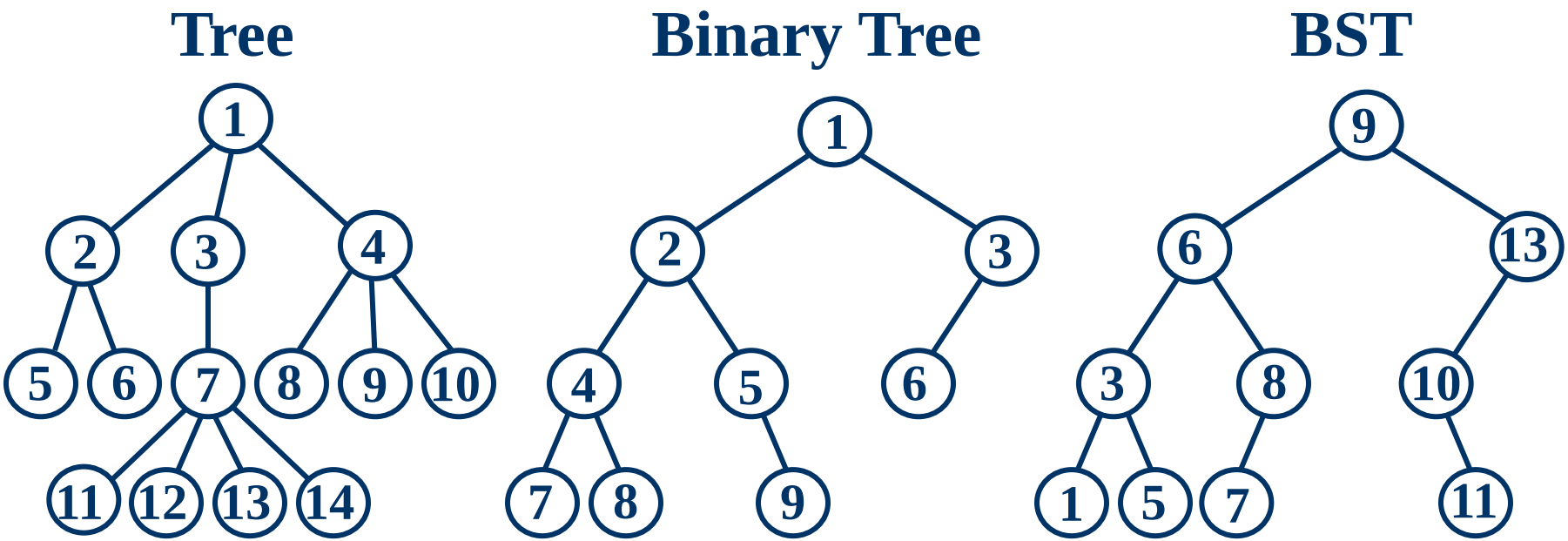

Data Type
ADT:
•Type
•Operations
Data Items:
Logical Form
DS:
•Storage Space
•Subroutines
Data Items:
Physical Form

Classification of Logical Form

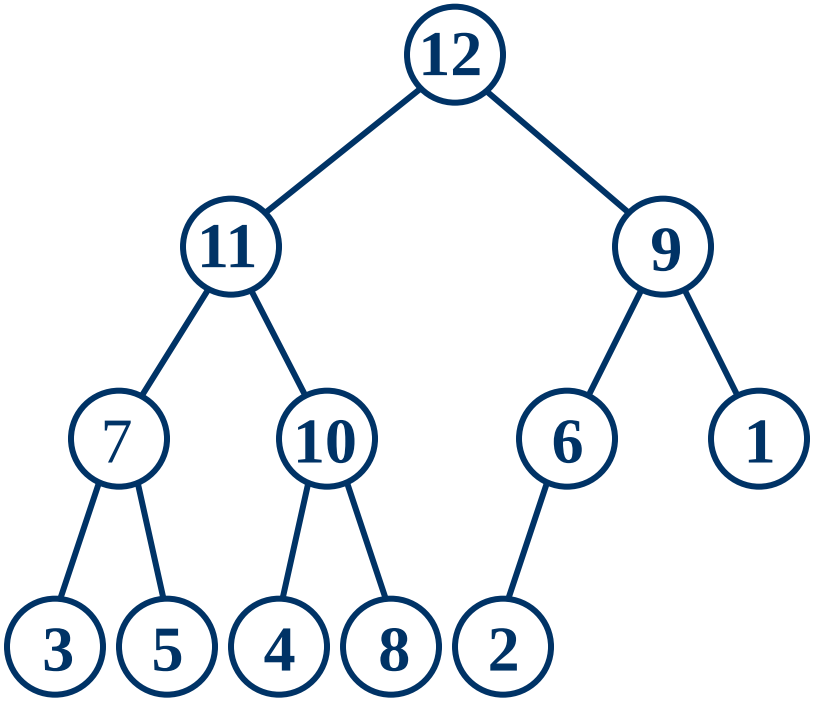
- **Linear Structure**
  - **Linear List**
- **Non-Linear Structure**
  - **Multiple Dimensional Array**
  - **List**
  - **Tree**
  - **Graph（or Network）**
- **Unstructured**
  - **Set**

Linear Structure

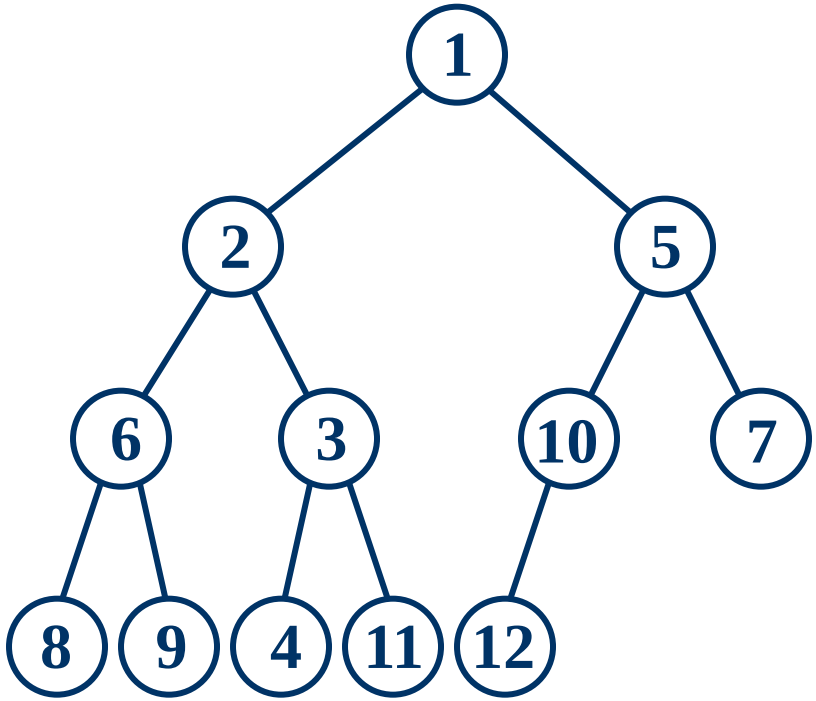
Tree Structure



Heap

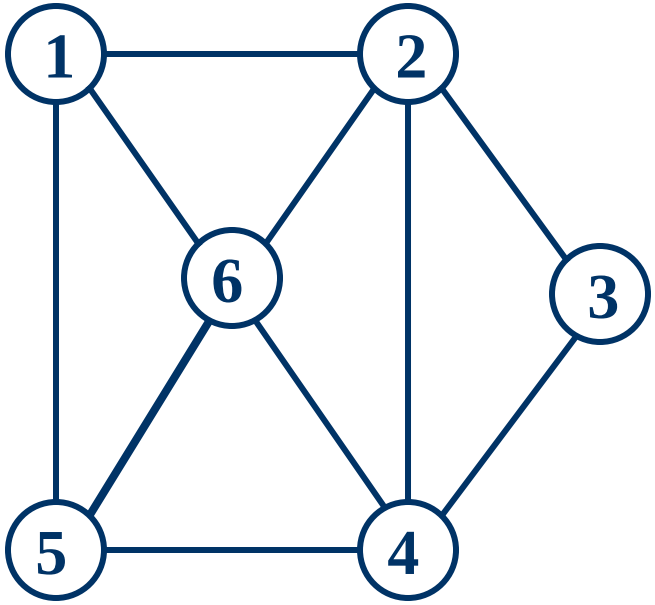


Max Heap

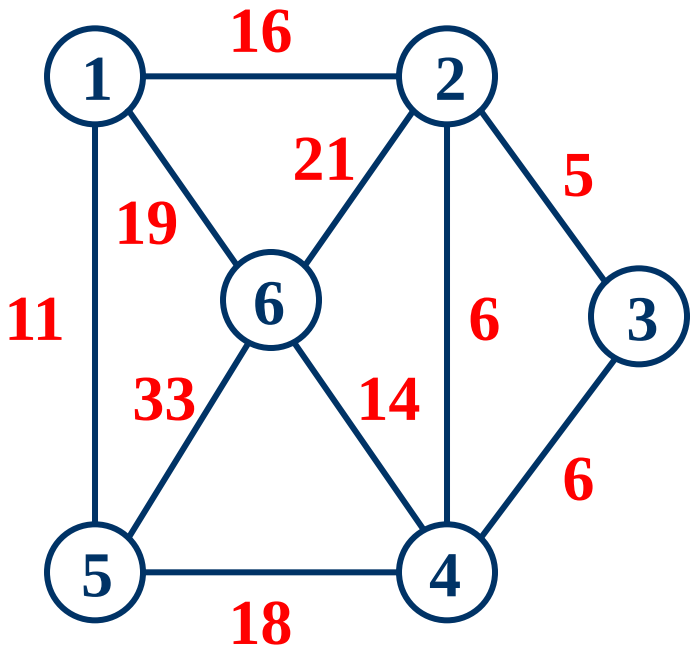


Min Heap

Graph



Network



Physical Form

- Logical structure realized in programming language，three task：
  - Content/Data
  - Relation
  - Additional space/ Temporary space
- Classification
  - Sequential storage representation
  - Linked storage representation
  - Index storage representation
  - Hash storage representation

Problems

- ***Problem*: a task to be performed.**
  - **Best thought of as inputs and matching outputs.**
  - **Problem definition should include constraints on the resources that may be consumed by any acceptable solution.**

Problems⇔mathematical functions

- A ***function*** is a matching between inputs (the domain) and outputs (the range).
- An ***input*** to a function may be single number, or a collection of information.
- The values making up an input are called the ***parameters*** of the function.
- A particular input must always result in the same output every time the function is computed.

Algorithms and Programs

- ***Algorithm*: a method or a process followed to solve a problem.**
- **An algorithm takes the input to a problem (function) and transforms it to the output.**
- **A problem can have many algorithms.**

An algorithm‘s properties

- **An algorithm possesses the following properties:**
  - **1. It must be correct.**
  - **2. It must be composed of a series of concrete steps.**
  - **3. There can be no ambiguity as to which step will be performed next.**
  - **4. It must be composed of a finite number of steps.**
  - **5. It must terminate.**

Computer program

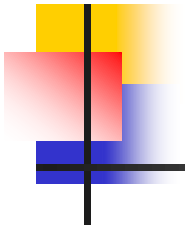
- **A computer program is an instance, or concrete representation, for an algorithm in some programming language.**

Estimation Techniques

- **Determine the major parameters that affect the problem.**
- **Derive an equation that relates the parameters to the problem.**
- **Select values for the parameters, and apply the equation to yield an estimated solution.**

Asymptotic Performance

- **In this course, we care most about *asymptotic performance***
  - **How does the algorithm behave as the problem size gets very large?**
    - **Running time**
    - **Memory/storage requirements**
    - **Bandwidth/power requirements/logic gates/etc.**



Asymptotic Notation

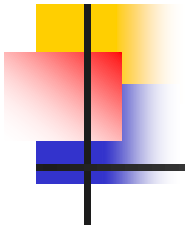
- **By now you should have an intuitive feel for asymptotic (big-O) notation:**
  - ***What does O(n) running time mean? O(n^2)? O(n lg n)?***
  - ***How does asymptotic running time relate to asymptotic memory usage?***
- **Our first task is to define this notation more formally and completely**

Input Size

- **Time and space complexity**
  - **This is generally a function of the input size**
    - **E.g., sorting, multiplication**
  - **How we characterize input size depends:**
    - **Sorting: number of input items**
    - **Multiplication: total number of bits**
    - **Graph algorithms: number of nodes & edges**
    - **Etc**

Running Time

- **Number of primitive steps that are executed**
  - **Except for time of executing a function call most statements roughly require the same amount of time**
    - **y = m * x + b**
    - **c = 5 / 9 * (t - 32)**
    - **z = f(x) + g(y)**
- **We can be more exact if need be**



Analysis

- **Worst case**
  - **Provides an upper bound on running time**
  - **An absolute guarantee**
- **Average case**
  - **Provides the expected running time**
  - **Very useful, but treat with care: what is “average”?**
    - **Random (equally likely) inputs**
    - **Real-life inputs**