第一章 绪论

- 什么是数据结构
- 抽象数据类型及面向对象概念
- 数据结构的抽象层次
- 用C++描述面向对象程序
- 算法定义
- 模板
- 性能分析与度量
- 小结

“学生”表格

	学　号	姓　名	性别	籍　贯	出生年月
1	98131	刘激扬	男	北　京	1979.12
2	98164	衣春生	男	青　岛	1979.07
3	98165	卢声凯	男	天　津	1981.02
4	98182	袁秋慧	女	广　州	1980.10
5	98203	林德康	男	上　海	1980.05
6	98224	洪　伟	男	太　原	1981.01
7	98236	熊南燕	女	苏　州	1980.03
8	98297	宫　力	男	北　京	1981.01
9	98310	蔡晓莉	女	昆　明	1981.02
10	98318	陈　健	男	杭　州	1979.12

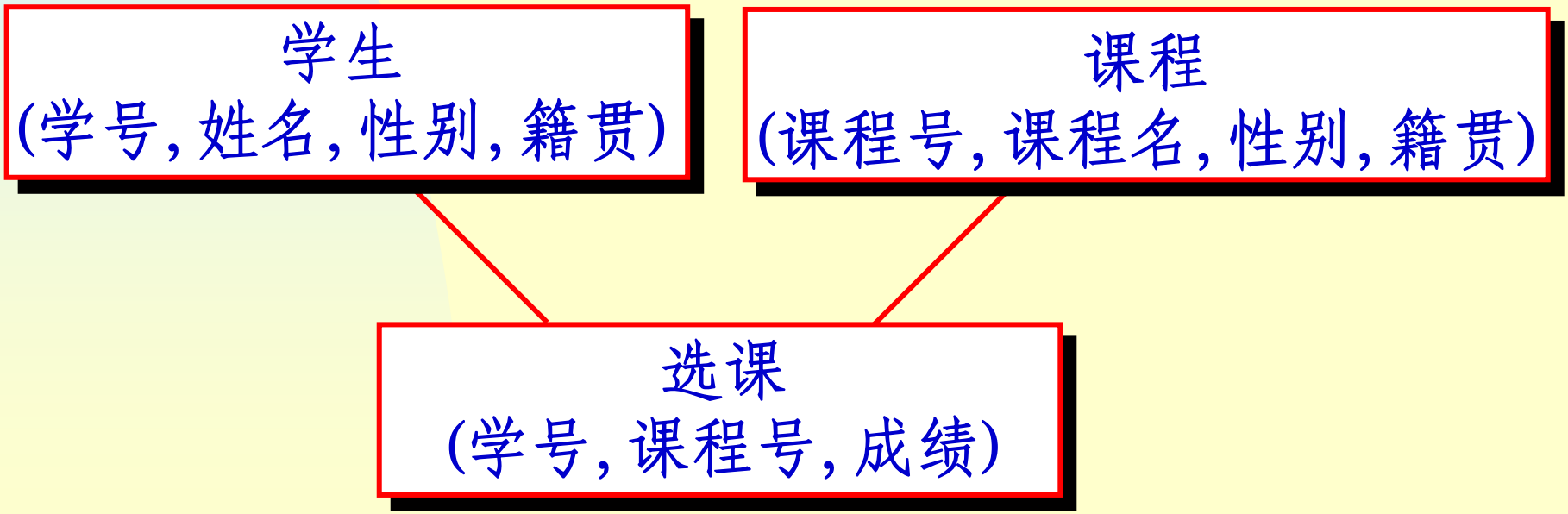
“课程”表格

课程编号	课 程 名	学时
024002	程序设计基础	64
024010	汇编语言	48
024016	计算机原理	64
024020	数据结构	64
024021	微机技术	64
024024	操作系统	48
024026	数据库原理	48

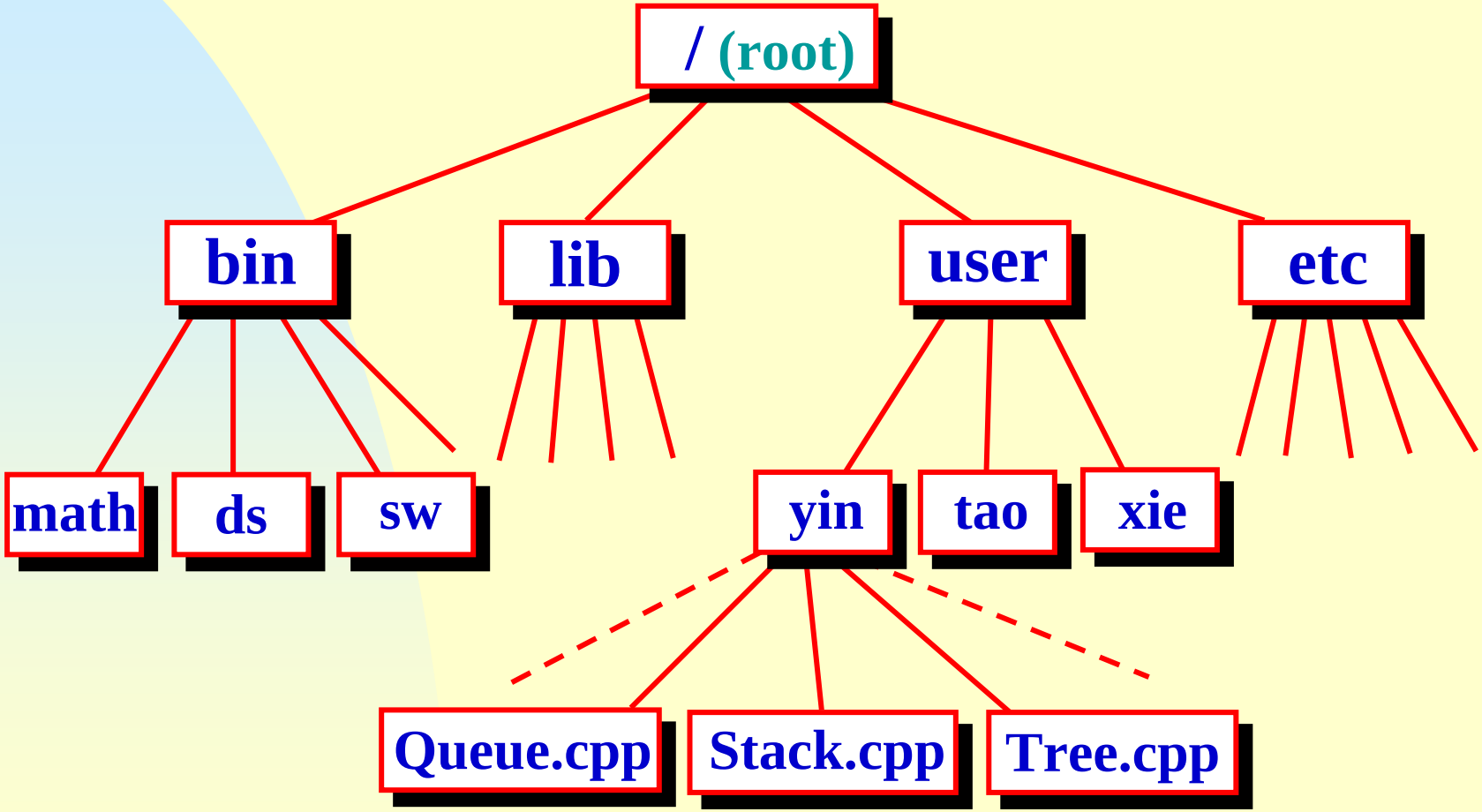
选课单包含如下信息

学号　课程编号　成绩　时间

学生选课系统中实体构成的网状关系



UNIX文件系统的系统结构图



- **数据**：数据是信息的载体，是描述客观事物的数、字符、以及所有能输入到计算机中，被计算机程序识别和处理的符号的集合。
  - 数值性数据
  - 非数值性数据
- **数据对象**：数据的子集。具有相同性质的数据成员（数据元素）的集合。
  - 整数数据对象 $N = \{ 0, \pm1, \pm2, \ldots \}$
  - 学生数据对象

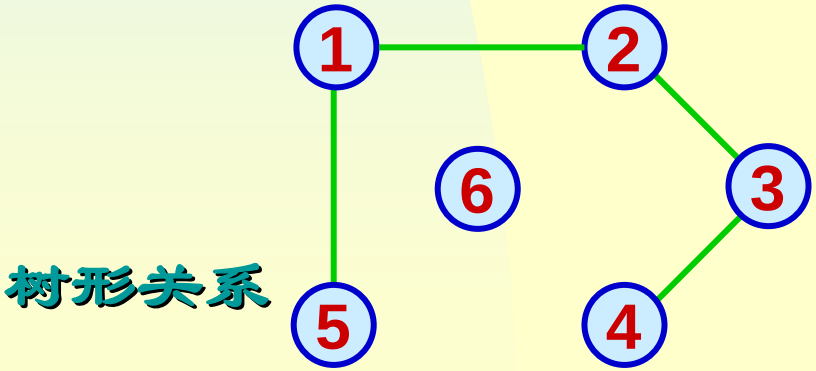
什么是数据结构

定义：由某一数据对象及该对象中所有数据成员之间的关系组成。记为：

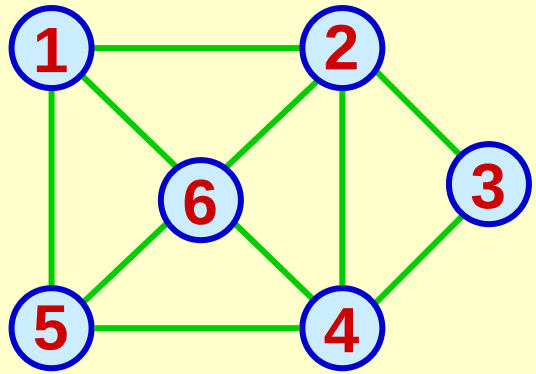
Data_Structure = {D, R}

其中，D是某一数据对象，R是该对象中所有数据成员之间的关系的有限集合。

*n*个网站之间的连通关系



树形关系



网状关系

抽象数据类型及面向对象概念

- 数据类型

  定义：一组性质相同的值的集合，以及定义于这个值集合上的一组操作的总称。

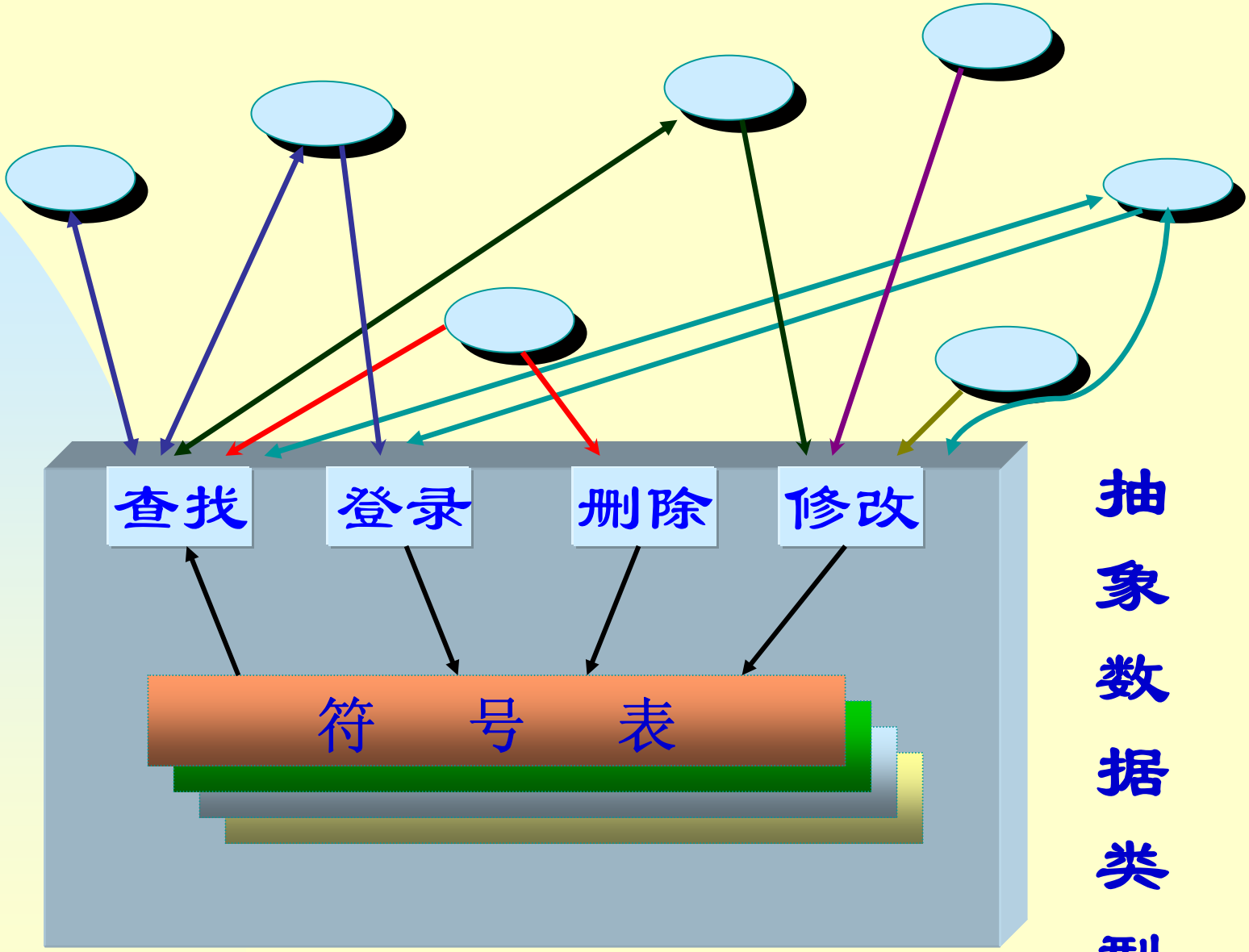
- C语言中的数据类型

  char　int　float　double　void

  字符型 整型 浮点型 双精度型 无值

抽象数据类型
(*ADTs: Abstract Data Types*)

- 由用户定义，用以表示应用问题的数据模型
- 由基本的数据类型组成，并包括一组相关的服务（或称操作）
- 信息隐蔽和数据封装，使用与实现相分离



抽象数据类型

自然数的抽象数据类型定义

ADT ***NaturalNumber*** **is**

objects: 一个整数的有序子集合,它开始于0,
结束于机器能表示的最大整数(*MaxInt*)。

Function: 对于所有的 $x, y \in$ *NaturalNumber*;
False, *True* $\in$ *Boolean*, +、-、<、==、=等都是可用的服务。

Zero( ) : 返回自然数0

NaturalNumber

IsZero(x) :	if (*x*==0) 返回*True*
Boolean	else 返回*False*
Add (x, y) :	if (*x*+*y*<=*MaxInt*)返回 *x*+*y*
NaturalNumber	else 返回*MaxInt*
Subtract (x, y) :	if (*x* < *y*) 返回 0
NaturalNumber	else 返回 *x* - *y*
Equal (x, y) :	if (*x*==*y*) 返回*True*
Boolean	else 返回 *False*
Successor (x) :	if (*x*==*MaxInt*) 返回 *x*
NaturalNumber	else 返回 *x*+1

end *NaturalNumber*

面向对象的概念

面向对象 = 对象 + 类 + 继承 + 通信

- 对象
  - 在应用问题中出现的各种实体、事件、规格说明等
  - 由一组属性值和在这组值上的一组服务（或称操作）构成
- 类 (class)，实例 (instance)
  - 具有相同属性和服务的对象归于同一类，形成类
  - 类中的一个对象为该类的一个实例

- 继承
  - 派生类：载重车，轿车，摩托车，…
    子类 特化类(特殊化类)
  - 基类：车辆
    父类 泛化类(一般化类)
- 通信
  - 消息传递

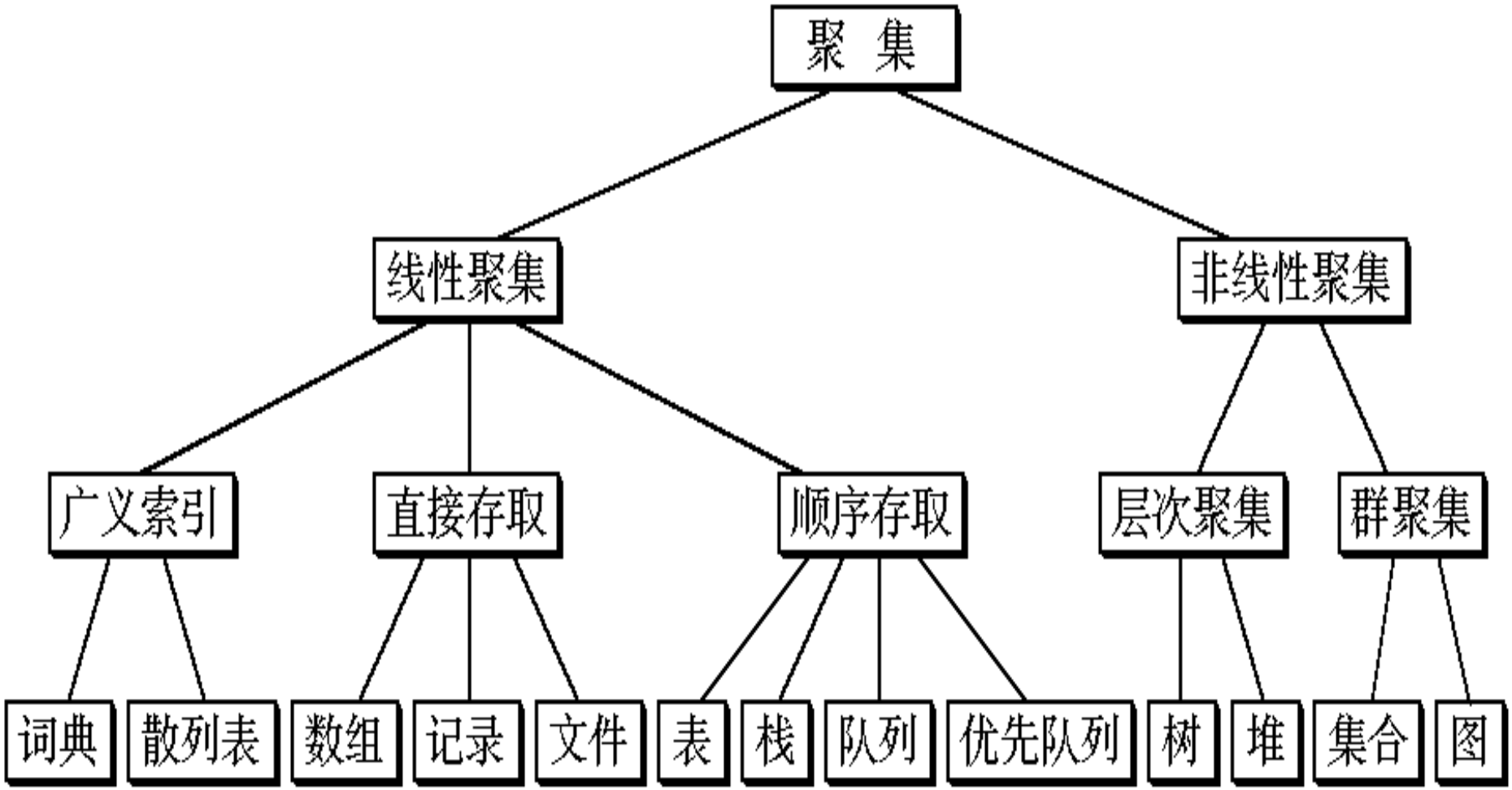
- 用于描述数据结构的语言

Smalltalk Effel C++ Java

数据结构的抽象层次

- 线性聚类
  - 直接存取类
  - 顺序存取类
  - 广义索引类
- 非线性聚类
  - 层次聚集类 树，二叉树，堆
  - 群聚集类 集合，图

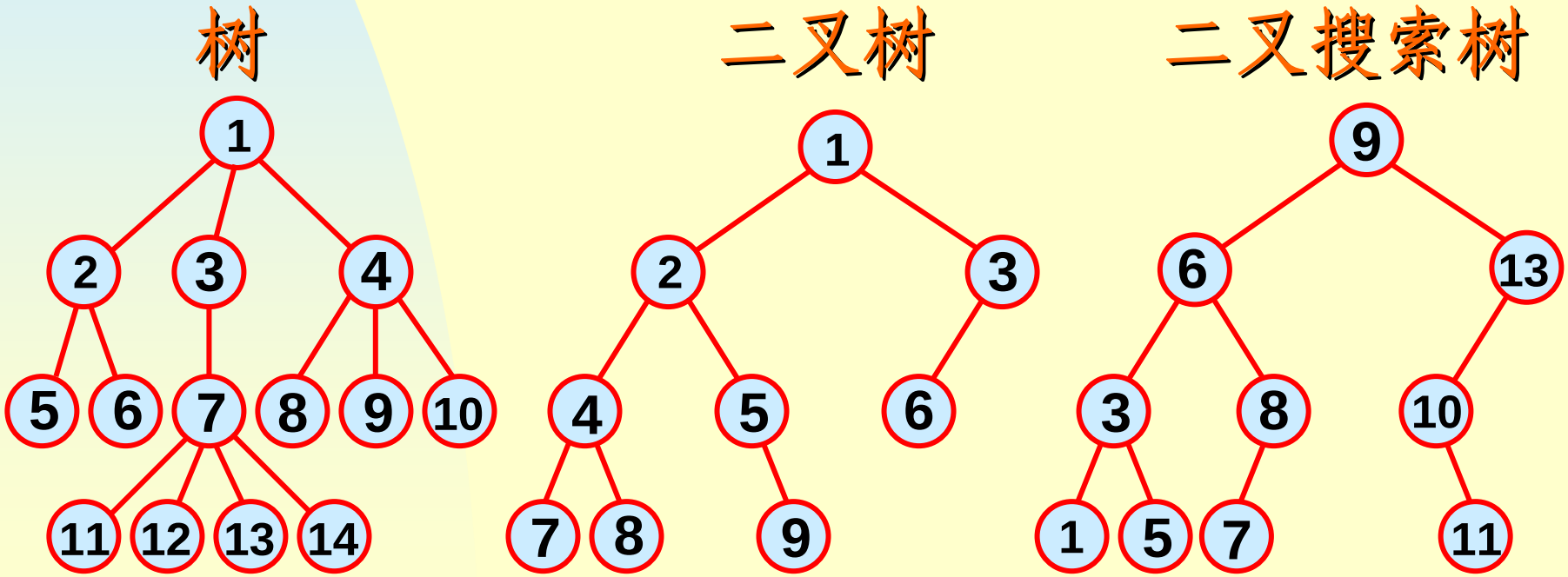
数据结构的抽象层次


聚 集
线性聚集
非线性聚集
广义索引
直接存取
顺序存取
层次聚集
群聚集
词典
散列表
数组
记录
文件
表
栈
队列
优先队列
树
堆
集合
图

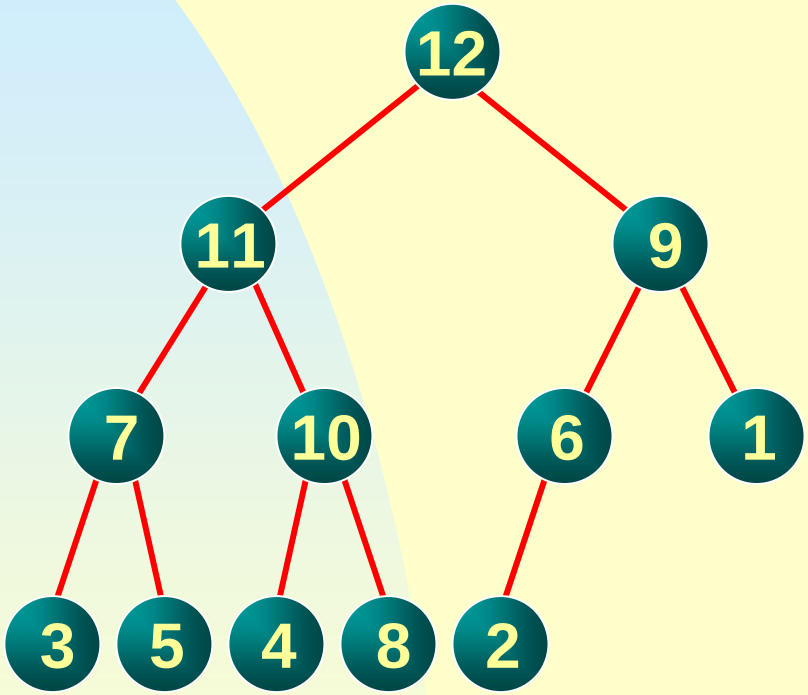
线性关系


bin
dev
etc
lib
user

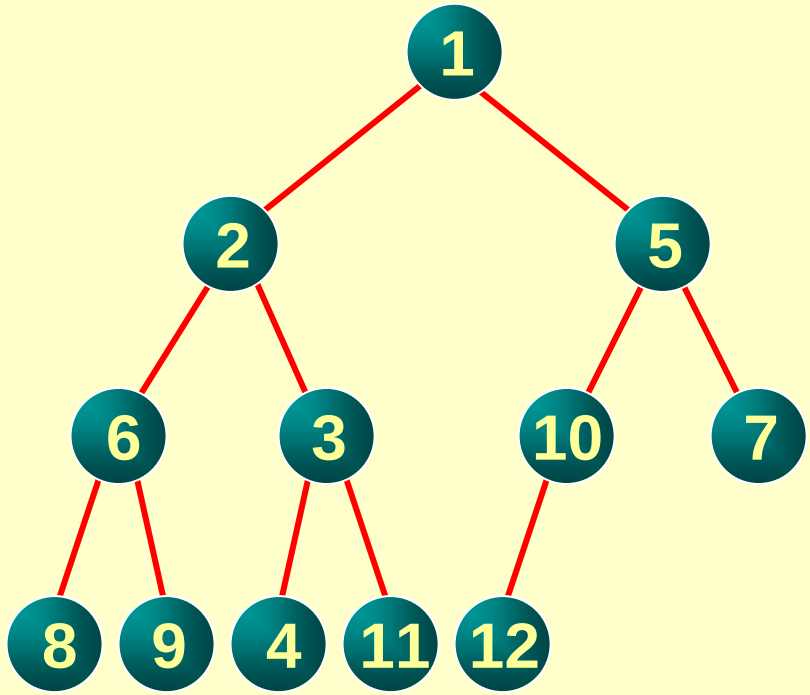
树形结构


树
二叉树
二叉搜索树
1
2
3
4
5
6
7
8
9
10
11
12
13
14
1
2
3
4
5
6
7
8
9
9
6
13
3
8
10
1
5
7
11

堆结构

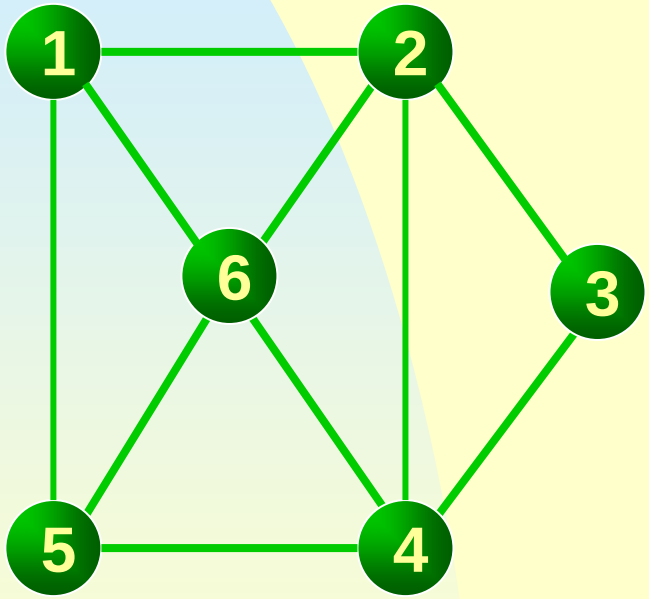


“最大”堆

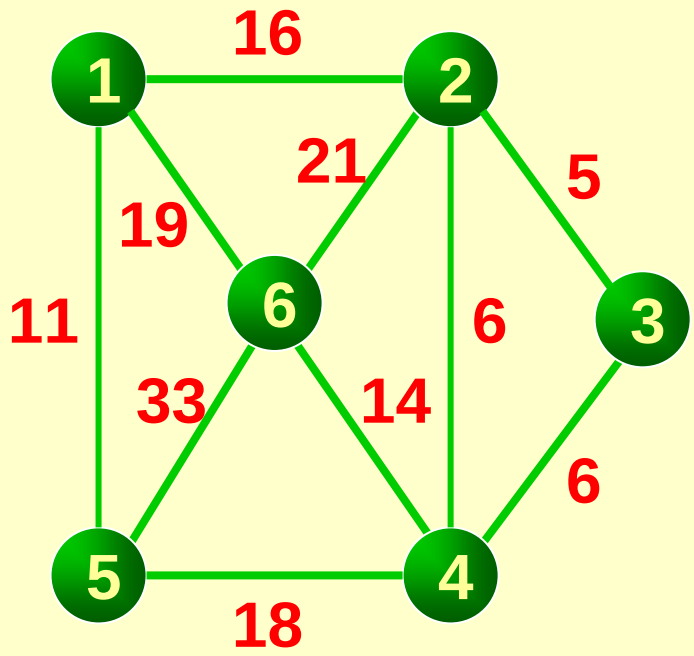


“最小”堆

群聚类



图结构



网络结构

算法定义

- 定义：一个有穷的指令集，这些指令为解决某一特定任务规定了一个运算序列
- 特性：
  - 输入 有0个或多个输入
  - 输出 有一个或多个输出(处理结果)
  - 确定性 每步定义都是确切、无歧义的
  - 有穷性 算法应在执行有穷步后结束
  - 有效性 每一条运算应足够基本

算法设计 自顶向下，逐步求精

- 事例学习：选择排序问题
- 明确问题：非递减排序
- 解决方案：逐个选择最小数据
- 算法框架：

```
for ( int i = 0; i < n-1; i++ ) {    //n-1趟
    从a[i]检查到a[n-1];
    若最小的整数在a[k], 交换a[i]与a[k];
}
```

- 细化程序：程序 *SelectSort*

```
void selectSort ( int a[ ], const int n ) {
//对n个整数a[0],a[1],…,a[n-1], 按非递减顺序排序
    for ( int i = 0; i < n−1; i++ ) {
        int k = i;
        //从a[i]检查到a[n-1], 找最小的整数, 在a[k]
        for ( int j = i+1; j < n; j++ )
            if ( a[j] < a[k] ) k = j;
                //k指示当前找到的最小整数
        int temp = a[i];  a[i] = a[k];   a[k] = temp;
                //交换a[i]与a[k]
    }
}
```

模板 (template)

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

用模板定义用于排序的数据表(*dataList*)类

```
#ifndef DATALIST_H
#define DATALIST_H
#include <iostream.h>
template <class Type>
class dataList {
private:
    Type *Element;
    int ArraySize;
    void Swap (const int m1, const int m2);
    int MaxKey (const int low, const int high);
```

```
public:
    dataList (int size = 10) : ArraySize (size),
        Element (new Type [Size]) { }
    ~dataList ( ) {delete [ ] Element;}
    void Sort ( );
    friend ostream& operator << (ostream&
        outStream, const datalist<Type>&
        outList);
    friend istream& operator >> (istream&
        inStream, const datalist<Type>&
        inList);
};
#endif
```

*dataList*类中所有操作作为模板函数的实现

```
#ifndef SELECTTM_H
#define SELECTTM_H
#include "datalist.h"
template <class Type>
void dataList <Type>::
Swap (const int m1,  const int m2) {
//交换由m1, m2为下标的两个数组元素的值
    Type temp = Element [m1];
    Element [m1] = Element [m2];
    Element [m2] = temp;
}
```

```
template <class Type>
int dataList<Type>::
MaxKey (const int low, const int high) {
//查找数组Element[low] ~ Element[high]中
//的最大值，函数返回其位置
    int max = low;
    for (int k = low+1, k <= high, k++)
        if ( Element[max] < Element[k] )
            max = k;
    return max;
}
```

```
template <class Type>
ostream&operator<< (ostream& OutStream,
  const dataList<Type> OutList) {
    OutStream << "Array Contents : \n";
    for (int i = 0; i < OutList.ArraySize; i++)
        OutStream << OutList.Element[i] << ' ';
    OutStream << endl;
    OuStream << "Array Current Size : " <<
        OutList.ArraySize << endl;
    return OutStream;
}
```

```
template <class Type>
istream& operator >> (istream& InStream,
    dataList<Type> InList) {
//输入对象为InList，输入流对象为InStream
    cout << "Enter array Current Size : ";
    Instream >> InList.ArraySize;
    cout << "Enter array elements : \n";
    for (int i = 0; i < InList.ArraySize; i++) {
        cout << "Elememt" << i << " :" ;
        InStream >> InList.Element[i];
    }
    return InStream;
}
```

```
template <class Type>
void dataList<Type>::Sort ( ) {
//按非递减顺序对ArraySize个关键码
//Element[0] ~Element[ArraySize-1]排序。
    for ( int i = ArraySize −1; i > 0; i−− ) {
        int j = MaxKey (0, i);
        if ( j != i ) swap (j, i);
    }
}
#endif
```

使用模板的选择排序算法的主函数

```
#include "selecttm.h"
const int SIZE = 10;
int main ( ) {
    dataList <int> TestList (SIZE);
    cin >> TestList;
    cout << "Testing Selection Sort : \n" <<
                TestList << endl;
    TestList.Sort ( );
    cout << "After sorting : \n" <<
                TestList << endl;
    return 0;
}
```

性能分析与度量

- 算法的性能标准
- 算法的后期测试
- 算法的事前估计

算法的性能标准

- 正确性
- 可使用性
- 可读性
- 效率
- 健壮性

算法的后期测试

在算法中的某些部位插装时间函数

time ( )

测定算法完成某一功能所花费的时间

顺序搜索 (*Sequenial Search*)

```
行   int seqsearch ( int a[ ], const int n,
          const int x )        //a[0],…,a[n−1]
1    {
2        int i = 0;
3        while ( i < n && a[i] != x )
4            i++;
5        if ( i == n ) return −1;
6        return i;
7    }
```

插装 time( ) 的计时程序

```
void TimeSearch ( )
{  //在1..1000中搜索0,10,…,90,100,200,…,1000
   int a[1001], n[20];
   for ( int j = 1; j <= 1000; j++ )
        a[j] = j;              //a[1]=1, a[2]=2, …
   for ( j = 0; j < 10; j++ ) {
        n[j] = 10*j;   //n[0]=0, n[1]=10, n[2]=20
        n[j+10] = 100*( j+1 );
   }              //n[10]=100, n[11]=200, n[12]=300 ...
   cout << " n     time" << endl;
```

```
    for ( j = 0; j < 20; j++ ) {
        long start, stop;
        time (&start);
        int k = seqsearch (a, n[j], 0);
        time (&stop);
        long runTime = stop - start;
        cout << " " << n[j] << " " <<
                        runTime << endl;
    }
    cout << "Times are in hundredths of a
                    second." << endl;
}
```

程序测试结果输出

n	0	10	20	30	40	50	60	70	80	90
run Time	0	0	0	0	0	0	0	0	0	0

n	100	200	300	400	500	600	700	800	900	1000
run Time	0	0	0	0	0	0	0	0	0	0

改进的计时程序

```
void TimeSearch ( ) {
  int a[1001], n[20];
  const long r[20] = {300000, 300000, 200000,
     200000, 100000, 100000, 100000, 80000,
     80000, 50000, 50000, 25000, 15000, 15000,
     10000, 7500, 7000, 6000, 5000, 5000 };
  for ( int j = 1; j <= 1000; j++ )
      a[j] = j;
  for ( j = 0; j < 10; j++ ) {
      n[j] = 10*j;  n[j+10] = 100*( j+1 );
  }
  cout << " n    totalTime    runTime" << endl;
```

```
for ( j = 0; j < 20; j++ ) {
    long start, stop;
    time (&start);                          //开始计时
    for ( long b = 1; b <= r[j]; b++ )
        int k = seqsearch(a, n[j], 0 ); //执行r[j]次
    time (&stop);                           //停止计时
    long totalTime = stop − start;
    float runTime =
        (float)(totalTime) / (float)(r[j]);
    cout << " " << n[j] << " " << totalTime
         << " " << runTime << endl;
  }
}
```

程序测试结果输出

n	0	10	20	30	40	50	60	70	80	90
总的运行时间	241	533	582	736	467	565	659	604	681	472
单个运行时间	0.000008	0.000018	0.000029	0.000037	0.000047	0.000056	0.000066	0.000075	0.000085	0.0094

n	100	200	300	400	500	600	700	800	900	1000
总的运行时间	527	505	451	593	494	439	484	467	434	484
单个运行时间	0.000505	0.000502	0.000301	0.000395	0.000394	0.000285	0.000291	0.000278	0.000268	0.000268

算法的事前估计

- 空间复杂度
- 时间复杂度

空间复杂度度量

- **存储空间的固定部分**
  程序指令代码的空间，常数、简单变量、定长成分(如数组元素、结构成分、对象的数据成员等)变量所占的空间
- **可变部分**
  尺寸与实例特性有关的成分变量所占空间、引用变量所占空间、递归栈所用的空间、通过**new**和**delete**命令动态使用的空间

时间复杂度度量

- 编译时间
- 运行时间
  - 程序步
    - 语法上或语义上有意义的一段指令序列
    - 执行时间与实例特性无关
    - 例如：

      注释：程序步数为0

      声明语句：程序步数为0

      表达式：程序步数为1

- 程序步确定方法
  - 插入计数全局变量*count*
  - 建表，列出个语句的程序步

例 以迭代方式求累加和的函数

```
行   float sum ( float a[ ], const int n )
 1   {
 2       float s = 0.0;
 3       for ( int i = 0; i < n; i++ )
 4           s += a[i];
 5       return s;
 6   }
```

在求累加和程序中加入count语句

```
float sum ( float a[ ], const int n ) {
    float s = 0.0;
    count++;        //count统计执行语句条数
    for ( int i = 0; i < n; i++ ) {
        count++;    //针对for语句
        s += a[i];
        count++;
    } //针对赋值语句
    count++;        //针对for的最后一次
    count++;        //针对return语句
    return s;
}
```

执行结束得 程序步数 $count = 2 * n + 3$

程序的简化形式

```
void sum ( float a[ ], const int n )
{
    for ( int i = 0; i < n; i++ )
        count += 2;
    count += 3;
}
```

注意：

一个语句本身的程序步数可能不等于该语句一次执行所具有的程序步数。

例如：赋值语句

$x = sum(R, n);$

本身的程序步数为1;

一次执行对函数 $sum(R, n)$ 的调用需要的程序步数为 $2*n+3$;

一次执行的程序步数为

$1+2*n+3 = 2*n+4$

计算累加和程序
程序步数计算工作表格

程 序 语 句	一次执行所需程序步数	执行频度	程序步数
{	0	1	0
float *s* = 0.0;	1	1	1
for (**int** *i*=0; *i*<*n*; *i*++)	1	*n*+1	*n*+1
s += *a*[*i*];	1	*n*	*n*
return *s*;	1	1	1
}	0	1	0
	总程序步数		2*n*+3

时间复杂度的渐进表示法

- 大O表示法
- 加法规则　针对并列程序段

$$T(n, m) = T1(n) + T2(m)$$
$$= O(\max(f(n), g(m)))$$

$$c < \log_2 n < n < n\log_2 n < n^2 < n^3 < 2^n < 3^n < n!$$

- 乘法规则　针对嵌套程序段

$$T(n, m) = T1(n) * T2(m)$$
$$= O(f(n) * g(m))$$

- 渐进的空间复杂度

$$S(n) = O(f(n))$$

两个并列循环的例子

```
void example (float x[ ][ ], int m, int n, int k) {
    float sum [ ];
    for ( int i = 0; i < m; i++ ) {  //x[ ][ ]中各行
        sum[i] = 0.0;                //数据累加
        for ( int j=0; j<n; j++ ) sum[i] += x[i][j];
    }
    for ( i = 0; i < m; i++ )   //打印各行数据和
        cout << "Line " << i <<
                        " : " <<sum [i] << endl;
}
```

渐进时间复杂度为 **O(*max* (*m**n, m))**

```
template <class Type>    //起泡排序的算法
void dataList<Type>::bubbleSort ( ) {
//对表 Element[0] 到 Element[ArraySize-1]
//逐趟进行比较, ArraySize 是表当前长度
     int i = 1;  int exchange = 1;
        //当exchange为0则停止排序
     while ( i < ArraySize && exchange ) {
        BubbleExchange ( i, exchange );
        i++;
    } //一趟比较
}
```

```
template <class Type>
void dataList<Type>::
BubbleExchange(const int i, int & exchange ){
    exchange = 0;              //假定元素未交换
    for ( int j = ArraySize−1; j>=i; j−−)
        if ( Element[j−1] > Element[j] ) {
            Swap ( j −1, j );  //发生逆序
                  //交换Element[j-1]与Element[j]
            exchange = 1;    //做“发生了交换”标志
        }
}
```

渐进时间复杂度

$$O(f(n)*g(n)) = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2}$$

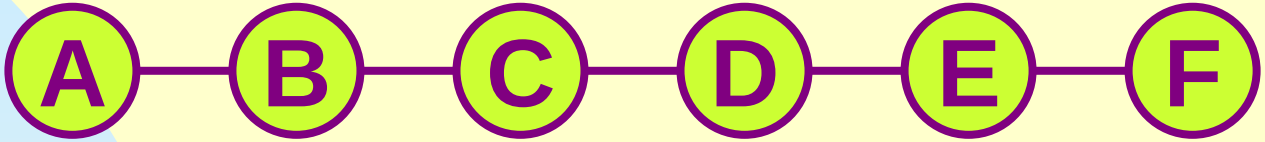
小结 为什么要学习数据结构?

- 它研究了计算机需要处理的数据对象和对象之间的关系。
- 它刻画了应用中涉及到的数据的逻辑组织。
- 它描述了数据在计算机中如何存储、传送、转换。

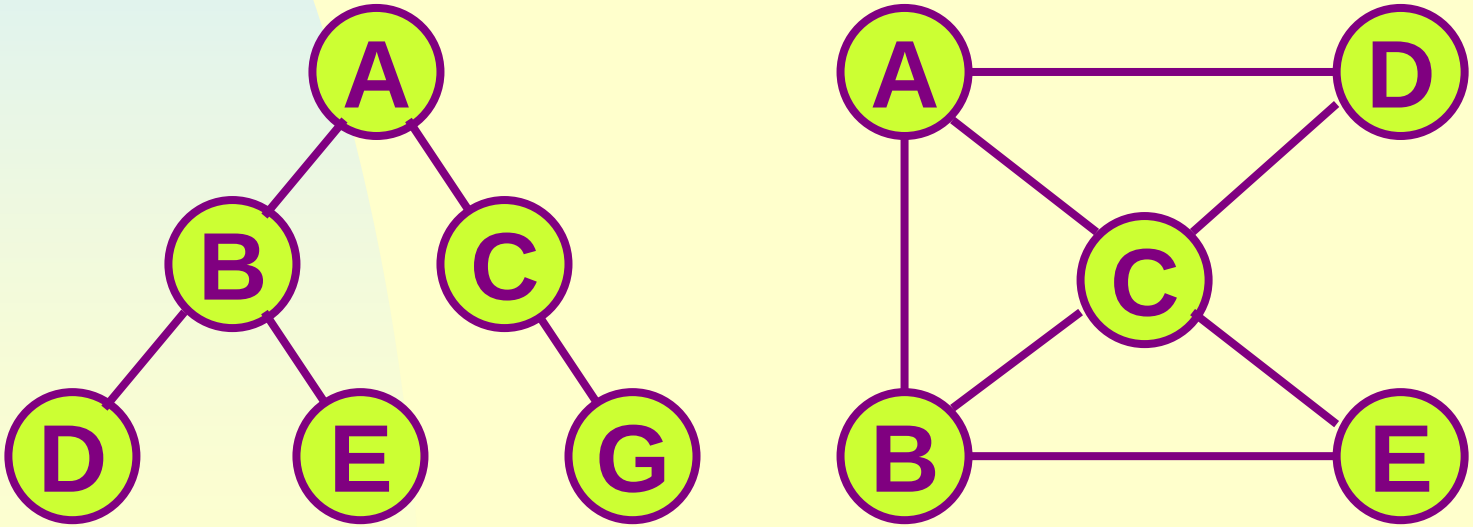
主要讨论哪几种数据结构?

- 从传统的观点来看
  - 数据的逻辑结构
    - 线性结构
    - 非线性结构

线性结构



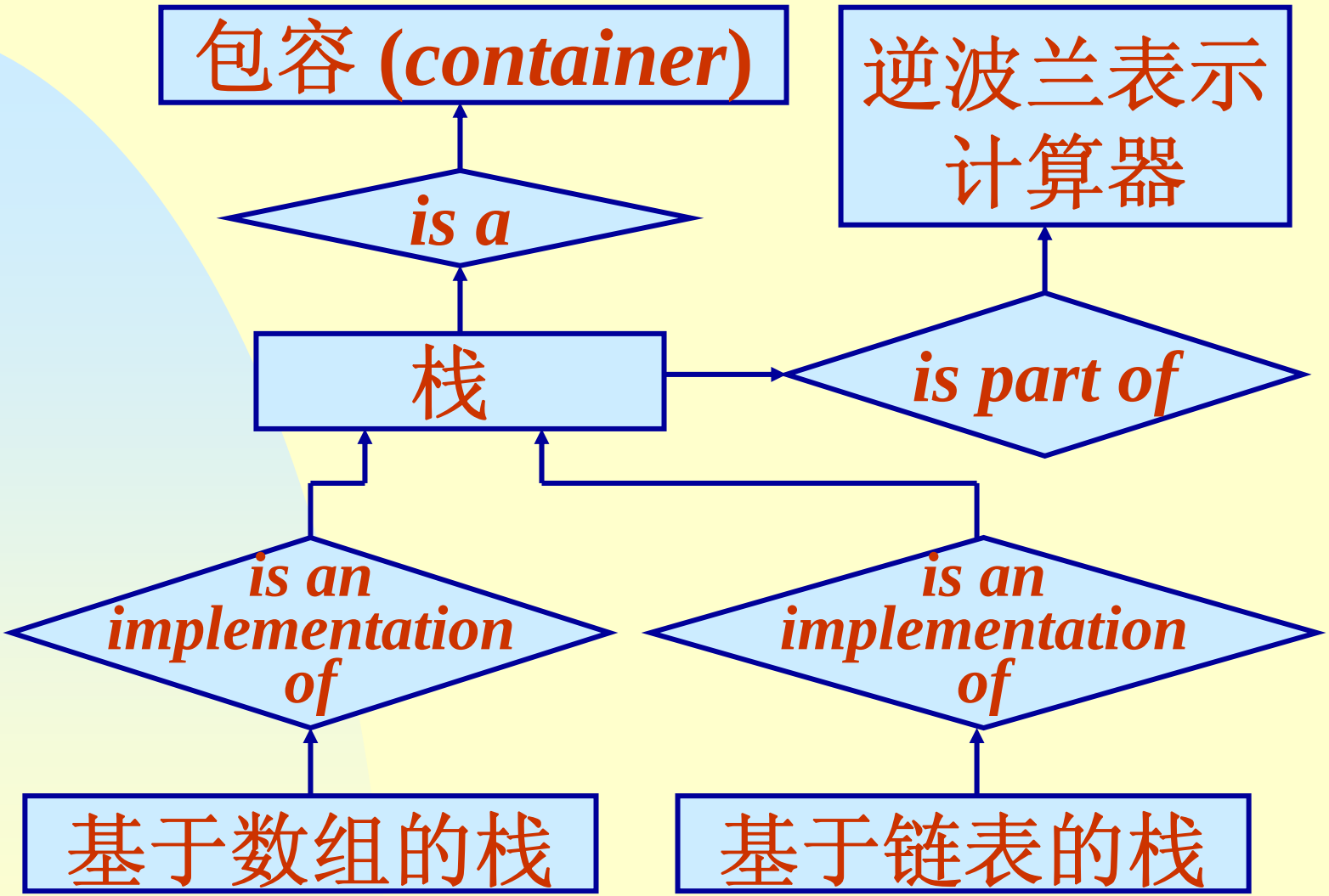
非线性结构



- 数据的物理结构
  - 顺序结构
  - 链表结构
  - 散列结构
  - 索引结构
- 在该数据结构上的操作

- 从面向对象的观点来看
  - 应用级的类与关系
    - 分类　一般与特殊
    - 组装　整体与部分
    - 关联　包容与实体
  - 实现级的类与关系

- 消息　消息通信 *refers to*
- 组装　一个类的实例是另一个类的实现部分 *is part of*
- 继承　从既存类定义新类
  - *is a*
  - *is kind of*
  - *is an implementation of*


包容 (container)
逆波兰表示计算器
is a
栈
is part of
is an implementation of
is an implementation of
基于数组的栈
基于链表的栈

为什么选用面向对象及C++语言讲述数据结构?

- PASCAL与C描述是面向过程的。
- C++描述兼有面向过程与面向对象的特点。
- Java描述是面向对象的。
- 用面向对象及C++描述与国际接轨，是市场需要

学习数据结构需要注意些什么？

- 知识方面
  - 系统掌握基本数据结构的特点及其不同实现。
  - 了解并掌握各种数据结构上主要操作的实现及其性能（时间、空间）的分析。

- 技能方面
  - 掌握各种数据结构的使用特性，在算法设计中能够进行选择。
  - 掌握常用的递归、回溯、迭代、递推等方法的设计
  - 掌握自顶向下、逐步求精的程序设计方法。

算法编制时需要注意些什么？

- 正确性

  前置（先决）条件、后置条件
- 可读性

  清晰性、一致性、简明性

  结构性、模块性
- 容错性

出错处理问题举例

- 试编写一个函数计算 $n!*2^n$ 的值，结果存放于数组 *A*[*arraySize*] 的第 *n* 个数组元素中 ($0 \le n \le arraySize$)
- 若设计算机中允许的整数的最大值为 *maxInt*，则当 $n > arraySize$ 或者对于某一个 k ($0 \le k \le n$)，使得 $k!*2^k > maxInt$ 时，应按出错处理。

可有如下三种不同的出错处理方式:

❶ 用 cerr << 及 exit (1) 语句来终止执行并报告错误;

❷ 用返回整数函数值 0，1 来实现算法，以区别是正常返回还是错误返回;

❸ 在函数的参数表设置一个引用型的整型变量来区别是正常返回还是某中错误返回。

```
#include "iostream.h"
#define arraySize 100
#define MaxInt 0x7fffffff

int calc ( int T[ ], int n ) {
   int i, value = 1;
   if ( n != 0 ) {
      for ( i = 1; i < n; i++) {
         value *= i * 2;
          if ( value > MaxInt / n / 2 )
              return 0;
      } //直接判断 i!*2^i > MaxInt 是危险的
```

```
        value *= n * 2;
    }
    T[n] = value;  return 1;  //n!*2^n → T[n]
}
void main ( ) {
    int A[arraySize], i;
    for ( i = 0; i < arraySize; i++ )
        if ( !calc ( A, i ) ) {
            cout << "failed at " << i << endl;
            break;
        }
}
```

第一章涉及的知识点

- 什么是数据与数据结构
- 抽象数据类型及面向对象概念
  - 数据类型;
  - 数据抽象与抽象数据类型;
  - 面向对象的概念
  - 用于描述数据结构的语言

- 数据结构的抽象层次
- 算法的性能分析与度量
  - 算法的性能标准;
  - 算法的后期测试;
  - 算法的事前估计;
  - 空间复杂度度量;
  - 时间复杂度度量;