

第一讲 基本概念

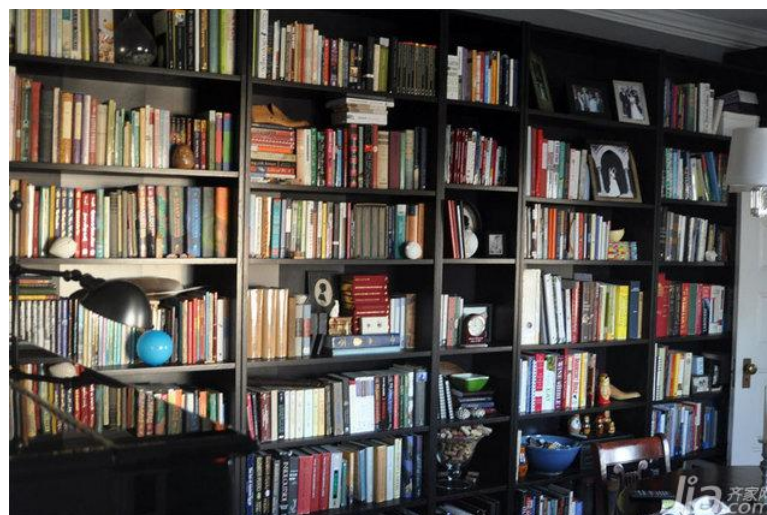
浙江大学 陈 越

1.1 什么是数据结构

官方统一定义——没有.....

- “数据结构是数据对象，以及存在于该对象的实例和组成实例的数据元素之间的各种联系。这些联系可以通过定义相关的函数来给出。”
 - Sartaj Sahni, 《数据结构、算法与应用》
- “数据结构是**ADT**（抽象数据类型 **Abstract Data Type**）的物理实现。”
 - Clifford A. Shaffer, 《数据结构与算法分析》
- “数据结构（**data structure**）是计算机中存储、组织数据的方式。通常情况下，精心选择的**数据结构**可以带来最优效率的**算法**。”
 - 中文维基百科

例1：如何在书架上摆放图书？



例1：如何在书架上摆放图书？



例1：如何在书架上摆放图书？

图书的摆放要使得**2**个相关操作方便实现：

- 操作**1**：新书怎么插入？
- 操作**2**：怎么找到某本指定的书？

例1：如何在书架上摆放图书？

■ 方法1：随便放

□ 操作1：新书怎么插入？

- 哪里有空放哪里，一步到位！

□ 操作2：怎么找到某本指定的书？

-累死



例1：如何在书架上摆放图书？

■ 方法2：按照书名的拼音字母顺序排放

□ 操作1：新书怎么插入？

- 新进一本《阿Q正传》

□ 操作2：怎么找到某本指定的书？

- 二分查找！



例1：如何在书架上摆放图书？

- 方法3：把书架划分成几块区域，每块区域指定摆放某种类别的图书；在每种类别内，按照书名的拼音字母顺序排放
 - 操作1：新书怎么插入？
 - 先定类别，二分查找确定位置，移出空位
 - 操作2：怎么找到某本指定的书？
 - 先定类别，再二分查找
 - 问题：空间如何分配？类别应该分多细？

解决问题方法的效率，
跟数据的组织方式有关

例2: 写程序实现一个函数PrintN, 使得传入一个正整数为N的参数后, 能顺序打印从1到N的全部正整数

```
void PrintN ( int N )
{ int i;
  for ( i=1; i<=N; i++ ) {
    printf("%d\n", i );
  }
  return;
}
```

循环实现

```
void PrintN ( int N )
{ if ( N ) {
    PrintN( N - 1 );
    printf("%d\n", N );
  }
  return;
}
```

递归实现

令 $N = 100, 1000, 10000, 100000, \dots$

例2：写程序实现一个函数PrintN，使得传入一个正整数为N的参数后，能顺序打印从1到N的全部正整数

```
#include <stdio.h>
void PrintN ( int N );
int main ()
{ int N;
  scanf ("%d", &N);
  PrintN( N );
  return 0;
}
```

```
10
1
2
3
4
5
6
7
8
9
10
Press any key to continue_
```

```
99977
99978
99979
99980
99981
99982
99983
99984
99985
99986
99987
99988
99989
99990
99991
99992
99993
99994
99995
99996
99997
99998
99999
100000
Press any key to continue_
```

```
100000
Press any key to continue_
```

递归实现



循环实现

解决问题方法的效率，
跟空间的利用效率有关

例3：写程序计算给定多项式在给定点 x 处的值

$$f(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} + a_nx^n$$

```
double f( int n, double a[], double x )
{ int i;
  double p = a[0];
  for ( i=1; i<=n; i++ )
    p += (a[i] * pow(x, i));
  return p;
}
```

$$f(x) = a_0 + x(a_1 + x(\cdots(a_{n-1} + x(a_n))\cdots))$$

```
double f( int n, double a[], double x )
{ int i;
  double p = a[n];
  for ( i=n; i>0; i-- )
    p = a[i-1] + x*p;
  return p;
}
```

clock(): 捕捉从程序开始运行到**clock()**被调用时所耗费的时间。这个时间单位是**clock tick**, 即“时钟打点”。

常数**CLK_TCK**(或**CLOCKS_PER_SEC**): 机器时钟每秒所走的时钟打点数。

```
#include <stdio.h>
#include <time.h>

clock_t  start, stop;
/* clock_t是clock()函数返回的变量类型 */
double  duration;
/* 记录被测函数运行时间, 以秒为单位 */
int main ()
{ /* 不在测试范围内的准备工作写在clock()调用之前*/
  start = clock(); /* 开始计时 */
  MyFunction();    /* 把被测函数加在这里 */
  stop = clock();  /* 停止计时 */
  duration = ((double)(stop - start))/CLK_TCK;
  /* 计算运行时间 */
  /* 其他不在测试范围的处理写在后面, 例如输出duration的值 */
  return 0;
}
```

例3：写程序计算给定多项式 $f(x) = \sum_{i=0}^9 i \cdot x^i$
在给定点 $x = 1.1$ 处的值 $f(1.1)$

```
double f1( int n, double a[], double x )
{ int i;
  double p = a[0];
  for ( i=1; i<=n; i++ )
    p += (a[i] * pow(x, i));
  return p;
}
```

```
double f2( int n, double a[], double x )
{ int i;
  double p = a[n];
  for ( i=n; i>0; i-- )
    p = a[i-1] + x*p;
  return p;
}
```



```

#include <stdio.h>
#include <time.h>
#include <math.h>
clock_t start, stop;
double duration;
#define MAXN 10 /* 多项式最大项数, 即多项式阶数+1 */
double f1( int n, double a[], double x );
double f2( int n, double a[], double x );

int main ()
{ int i;
  double a[MAXN]; /* 存储多项式的系数 */
  for ( i=0; i<MAXN; i++ ) a[i] = (double)i;

  start = clock();
  f1(MAXN-1, a, 1.1);
  stop = clock();
  duration = ((double)(stop - start))/CLK_TCK;
  printf("ticks1 = %f\n", (double)(stop - start));
  printf("duration1 = %6.2e\n", duration);

  start = clock();
  f2(MAXN-1, a, 1.1);
  stop = clock();
  duration = ((double)(stop - start))/CLK_TCK;
  printf("ticks2 = %f\n", (double)(stop - start));
  printf("duration2 = %6.2e\n", duration);

  return 0;
}

```

$$f(x) = \sum_{i=0}^9 i \cdot x^i$$

```

ticks1 = 0.000000
duration1 = 0.00e+000
ticks2 = 0.000000
duration2 = 0.00e+000
Press any key to continue

```

让被测函数重复运行充分多次，使得测出的总的时钟打点间隔充分长，最后计算被测函数平均每次运行的时间即可！

```
#include <stdio.h>
#include <time.h>
#include <math.h>

.....
#define MAXK 1e7 /* 被测函数最大重复调用次数 */
.....

int main ()
{
    .....

    start = clock();
    for ( i=0; i<MAXK; i++ ) /* 重复调用函数以获得充分多的时钟打点数*/
        f1(MAXN-1, a, 1.1);
    stop = clock();
    duration = ((double)(stop - start))/CLK_TCK/MAXK; /* 计算函数单次运行的时间 */
    printf("ticks1 = %f\n", (double)(stop - start));
    printf("duration1 = %6.2e\n", duration);

    .....

    return 0;
}
```

```
ticks1 = 10093.000000
duration1 = 1.01e-006
ticks2 = 1375.000000
duration2 = 1.38e-007
Press any key to continue
```

解决问题方法的效率，
跟算法的巧妙程度有关

所以到底什么是数据结构？ ？ ？

- **数据对象**在计算机中的组织方式
 - 逻辑结构
 - 物理存储结构
- 数据对象必定与一系列加在其上的**操作**相关联
- 完成这些操作所用的方法就是**算法**

抽象数据类型 (Abstract Data Type)

■ 数据类型

- 数据对象集
- 数据集合相关联的操作集

像java中会定义一个类，里面封装了成员变量和操作成员变量的方法，如常见的set/get方法

■ 抽象：描述数据类型的方法不依赖于具体实现

- 与存放数据的机器无关
- 与数据存储的物理结构无关
- 与实现操作的算法和编程语言均无关

只描述数据对象集和相关操作集“**是什么**”，并不涉及“**如何做到**”的问题

例4: “矩阵”的抽象数据类型定义

- 类型名称: 矩阵 (**Matrix**)

二维数组? 一维数组? 十字链表?

- 数据对象集: 一个 $M \times N$ 的矩阵 $A_{M \times N} = (a_{ij})$ ($i=1, \dots, M; j=1, \dots, N$)由 $M \times N$ 个三元组 $\langle a, i, j \rangle$ 构成, 其中 a 是矩阵元素的值, i 是元素所在的行号, j 是元素所在的列号。

- 操作集: 对于任意矩阵 $A, B, C \in \text{Matrix}$, 以及整数 i, j, M, N

- **Matrix Create**(**int** M , **int** N): 返回一个 $M \times N$ 的空矩阵;

- **int GetMaxRow**(**Matrix** A): 返回矩阵 A 的总行数;

- **int GetMaxCol**(**Matrix** A): 返回矩阵 A 的总列数;

- **ElementType GetEntry**(**Matrix** A , **int** i , **int** j): 返回矩阵 A 的第 i 行、第 j 列的元素; 抽象的类型: 例如java中的泛型, 如参数可以传入int/double/float...

- **Matrix Add**(**Matrix** A , **Matrix** B): 如果 A 和 B 的行、列数一致, 则返回矩阵 $C=A+B$, 否则返回错误标志;

- **Matrix Multiply**(**Matrix** A , **Matrix** B): 如果 A 的列数等于 B 的行数, 则返回矩阵 $C=AB$, 否则返回错误标志;

-

先按行加? 先按列加? 什么语言?