Computer Science Illuminated
SECOND EDITION
Nell Dale • John Lewis

Chapter 9

Abstract Data Types and Algorithms

Chapter Goals

- Define an abstract data type and discuss its role
- Distinguish between an array and a list
- Distinguish between an array-based implementation and a linked implementation
- Distinguish between a selection sort and a bubble sort
- Apply the selection sort, the bubble sort, and the Quicksort to a list of items by hand
- Apply the binary search algorithm
- Distinguish between the behavior of a stack and a queue

9-2

Abstract Data Types

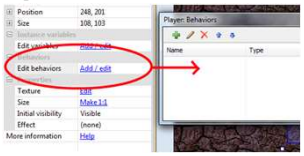
- **Abstract data type** A data type whose properties (data and operations) are specified independently of any particular implementation

The goal in design is to reduce complexity through abstraction

9-3

抽象数据类型实例

Sprite的方法与操作



Sprite隐藏了哪些复杂的东西？
例如：texture的存储，格式（信息隐藏）；
修改位置等产生的平移，旋转等数学变换。
（过程隐藏）；你定义player行为，它就受控制了，你不用知道如何实现（控制隐藏）

9-4

Abstract Data Types

- In computing, we view data from three perspectives
  - Application level
    - View of the data within a particular problem
  - Logical level
    - An abstract view of the data values (the domain) and the set of operations to manipulate them
  - Implementation level
    - A specific representation of the structure to hold the data items and the coding of the operations in a programming language

9-5

Abstract Data Types

- **Data structures** The implementation of a composite data fields in an abstract data type
- **Containers** Objects whole role is to hold and manipulate other objects

具体问题：一个班有最多有100学生，每个学生有学号、姓名、性别、身高、绩点。
逻辑视图：班，存学生的容器。容器提供三个操作，获取学生c.getNext()，是否存在学生c.exist()，重新迭代c.reset()。
学生是一个抽象数据结构类型，s.name 代表获取学生的姓名，s.height 代表获取学生的身高，等等
实现视图：用C语言结构体表示学生，学生类型数组表示班

9-6

案例：在逻辑视图上解决问题

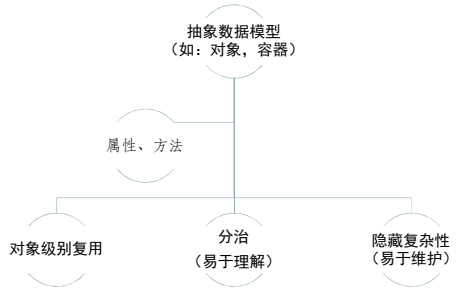
- 计算班上有多少女生

```
//input class
count = 0
class.reset()  //初始化容器内指针
WHILE class.exist() DO
   student = class.getNext()
   IF student.isFemale THEN count++
END WHILE
Output count
```

```
//简化的写法
//input class
count = 0
FOR EACH student in class DO
   IF student.isFemale THEN count++
END FOR
Output count
```

9-7

在逻辑层面抽象的意义



9-8

案例研究：图书馆找书

9-9

问题求解过程

- 识别问题中的对象和类
- 识别对象和类的
  - 属性
  - 行为
- 研究对象（类）之间的关系
  - 包含（part-of）
  - 继承（is-a）
- 由外向内逐步实现
  - 事件，行为

6-10

问题相关知识与应用场景

- 图书馆的藏书排架方法
  - 读者借书时一定会发现在图书的书脊上有一个标签,标签上有两组号码,上面的一组为分类号,下面的一组为书次号,这两组号码就构成了索书号.索书号是确定一本书排架的依据.
  - 索书号是D452.62/4052。“/”前面的部分是中国图书馆图书分类法的分类号。其中，“D”表示“政治、法律”大类，后面的数字是细分的小类。“/”后面的数字应该是表示这种书是该图书馆的“政治法律”类图书的第4052种。
- 在藏书架上查书的基本操作
  - 用书名查找图书
  - 用作者查找图书
  - 用索书号查找图书

9-11

对象 / 属性 / 方法

- 领域对象（domain objects）

  CLASS Bookshelf
  - book index：from-number, to-number
  - container: sorted books by book-index
  - Operations

  findByName（book-name）
  findByAuthor（author-name)
  findByIndex（book-index）

  CLASS Book
  - name
  - author
  - book-index

6-12

实现描述

```
Structure Book {
    string name
    string first-name，last-name
    string book-index
}
Array Bookshelf<Book>

Book findByName(String book_name){
    ……
}
……
```

9-13

实现过程描述（伪代码）

- 问题场景，你在书库门口，需要找你要的书（你手上有一个小字条，上面有书名、作者、索书号）。
- 请你根据你的经历，用自然语言描述找书过程。

9-14

算法与优化

- look for mybook 。（哪个好啊？）

```
算法1：sequence-search（mybook，bookshelfs）（顺序查找）：
FOR each bookshelf in bookshelfs DO
    IF mybook.index-number between bookshelf's form-number to to-number THEN
        return bookshelf.findByIndex(mybook.book-index)
    END IF
END FOR
```

```
算法2：binary-search（mybook, bookshelfs）（二分查找，书架是排序的）：
go to the middle bookshelf of the bookshelfs
IF mybook.index-number GREAT THEN to-number THEN
    binary-search（mybook, frontal bookshelfs）
END IF
IF mybook.index-number LESS THEN from-number THEN
    binary-search（mybook, posterior bookshelfs）
END IF
IF mybook.index-number BETWEEN form-number to to-number THEN
    return bookshelf.findByIndex(mybook.book-index)
END IF
```

9-15

算法与优化

- 对于计算机
  - 二分查找是最优的
  - 因为有最少的查找次数（最少的查找时间）
- 现实却是复杂的
  - 你可能希望走的距离最短
  - 你可能已经猜到了你要的书大致的位置
  - 可能你要的书被放乱了位置

9-16

Arrays

- An array is a named collection of homogeneous items in which individual items are accessed by their place within the collection
  - The place within the collection is called an *index*

Language	Array Declaration
Ada	`type Index_Range is range 1..10;` `type Ten_Things is array (Index_Range) of Integer;`
VB.NET	`Dim TenThings(10) As Integer`
C++/Java	`int tenThings[10];`

8-17

Arrays

[0]	1066
[1]	1492
[2]	1668
[3]	1945
[4]	1972
[5]	1510
[6]	999
[7]	1001
[8]	21
[9]	2001

Figure 8.8 Array variable *tenThings* accessed from *0..9*

8-18

Array-Based Implementations

- Recall that
  - an array is a named collection of homogeneous items
  - An item's place within the collection is called an index
- If there is no ordering on the items in the container, we call the container unsorted
- If there is an ordering, we call the container sorted

9-19

Array-Based Implementations

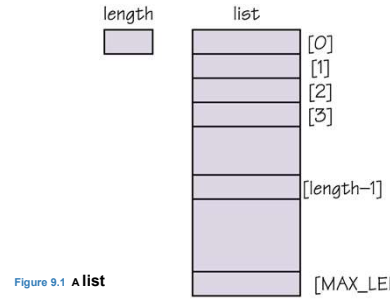


Figure 9.1 A list

9-20

Array-Based Implementations

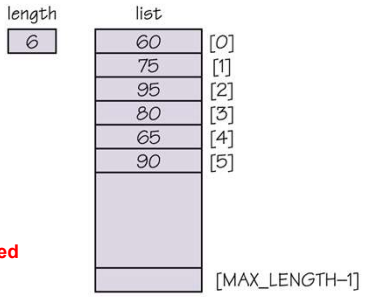


Figure 9.2
An unsorted list of integers

9-21

Array-Based Implementations

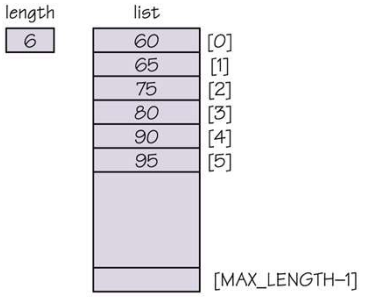


Figure 9.3
A sorted list of integers

9-22

Linked Implementation

- **Linked implementation** An implementation based on the *concept of a node*
- A node is made up of two pieces of information
  - the item that the user wants in the list, and
  - a pointer to the next node in the list

9-23

Linked Implementation

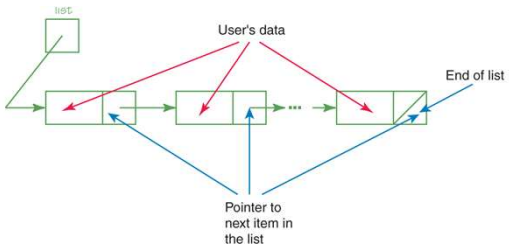


Figure 9.4 Anatomy of a linked list

9-24

Linked Implementation

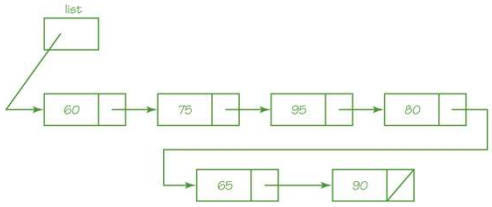


Figure 9.5 An unsorted linked list

9-25

Linked Implementation

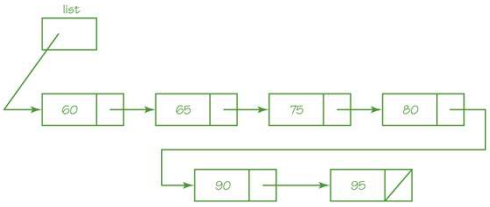


Figure 9.6 A sorted linked list

9-26

Linked Implementation

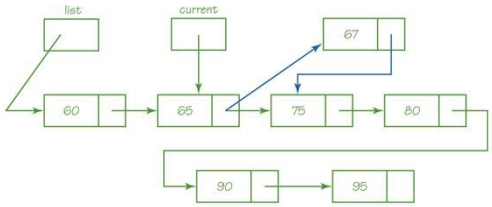


Figure 9.7 Store a node with *info* of 67 after *current*

9-27

Linked Implementation

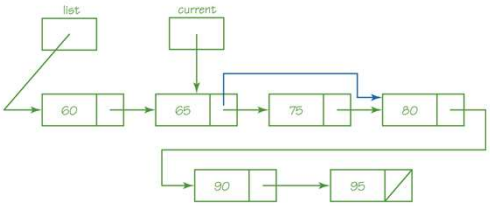


Figure 9.8 Remove node *next(current)*

9-28

容器：链表实现与数组实现对比

- 空间花费：数组少
- 按index访问数据：数组快
- 查找
  - 有序列表：数组有更好的查找算法
  - 无序列表：两者都只能顺序查找
- 插入数据：链表方便
- 删除数据：链表方便

9-29

Lists

- List operations
  - Create itself
  - Insert an item
  - Delete an item
  - Print itself
  - Know the number of items it contains
- **Generic data type** (or **class**) A data type or class in which the operations are specified but the type or class of the objects being manipulated is not

9-30

Sorting

- Because sorting a large number of elements can be extremely time-consuming, a good sorting algorithm is very desirable
- We present several quite different sorting algorithms

9-31

Selection Sort

- List of names
  - Put them in alphabetical order
    - Find the name that comes first in the alphabet, and write it on a second sheet of paper
    - Cross out the name on the original list
    - Continue this cycle until all the names on the original list have been crossed out and written onto the second list, at which point the second list is sorted

9-32

Selection Sort (cont.)

- A slight adjustment to this manual approach does away with the need to duplicate space
  - As you cross a name off the original list, a free space opens up
  - Instead of writing the minimum value on a second list, exchange it with the value currently in the position where the crossed-off item should go

9-33

Selection Sort

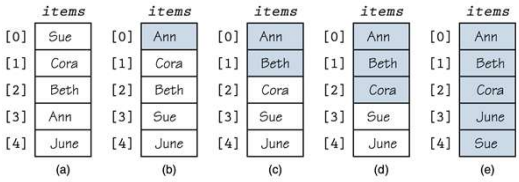


Figure 9.9 Example of a selection sort (sorted elements are shaded)

9-34

Select Sort 的算法描述

```
list = {Sue, Cora, Beth, Ann, June}
FOR i from 0 to list.length-2 DO
   Find index of the smallest item in list[i..list.length-1]
   IF index <> i THEN
      list.swap (i, index)
   END IF
END FOR
```

```
          0,    1,    2,    3,    4
list = {Sue, Cora, Beth, Ann, June}
FOR i from 0 to list.length-2 DO
   index = i
   FOR j from i+1 to list.length-1 DO
      IF list[j] < list[index] THEN index=j
   END FOR
   …
END FOR
```

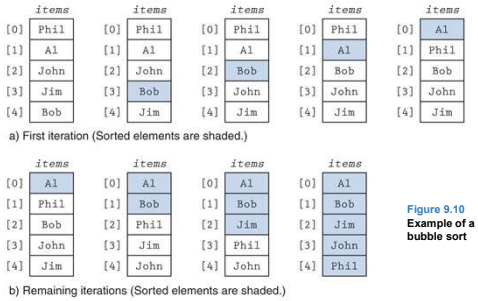
9-35

Bubble Sort

- A selection sort that uses a different scheme for finding the minimum value
  - Starting with the last list element, we compare successive pairs of elements, swapping whenever the bottom element of the pair is smaller than the one above it

9-36

Bubble Sort



a) First iteration (Sorted elements are shaded.)

b) Remaining iterations (Sorted elements are shaded.)

Figure 9.10
Example of a bubble sort

9-37

Bubble Sort 的算法描述

list = {Sue, Cora, Beth, Ann, June}

9-38

Quicksort

- Based on the idea that it is faster and easier to sort two small lists than one larger one
  - Given a large stack of final exams to sort by name
  - Pick a splitting value, say L, and divide the stack of tests into two piles, A–L and M–Z
  - note that the two piles do not necessarily contain the same number of tests
  - Then take the first pile and subdivide it into two piles, A–F and G–L
  - This division process goes on until the piles are small enough to be easily sorted by hand

9-39

Binary Search

- A **sequential search** of a list begins at the beginning of the list and continues until the item is found or the entire list has been searched
- A **binary search** looks for an item in a list using a divide-and-conquer strategy

9-40

Binary Search

- Binary Search Algorithm
  - Binary search algorithm assumes that the items in the list being searched are sorted
  - The algorithm begins at the middle of the list in a binary search
  - If the item for which we are searching is less than the item in the middle, we know that the item won't be in the second half of the list
  - Once again we examine the "middle" element (which is really the item 25% of the way into the list)
  - The process continues with each comparison cutting in half the portion of the list where the item might be

9-41

Binary Search

```
Boolean Binary Search (first, last)
If (first > last)
    return false
Else
    Set middle to (first + last)/ 2
    Set result to item.compareTo(list[middle])
    If ( result is equal to 0)
        return true
    Else
        If (result < 0)
            Binary Search (first, middle – 1)
        Else
            Binary Search (middle + 1, last)
```

9-42

Binary Search

[0]	ant
[1]	cat
[2]	chicken
[3]	cow
[4]	deer
[5]	dog
[6]	fish
[7]	goat
[8]	horse
[9]	camel
[10]	snake

Searching for cat

BinarySearch(0, 10)	middle: 5	cat < dog	
BinarySearch(0, 4)	middle: 2	cat < chicken	
BinarySearch(0, 1)	middle: 0	cat > ant	
BinarySearch(1, 1)	middle: 1	cat = cat	**Return: true**

Searching for zebra

BinarySearch(0, 10)	middle: 5	zebra > dog	
BinarySearch(6, 10)	middle: 8	zebra > horse	
BinarySearch(9, 10)	middle: 9	zebra > camel	
BinarySearch(10, 10)	middle: 10	zebra > snake	
BinarySearch(11, 10)		last > first	**Return: false**

Searching for fish

BinarySearch(0, 10)	middle: 5	fish > dog	
BinarySearch(6, 10)	middle: 8	fish < horse	
BinarySearch(6, 7)	middle: 6	fish = fish	**Return: true**

Figure 9.14 Trace of the binary search

9-43

Binary Search

Length	Sequential Search	Binary Search	
		Base 10	Base 2
10	5.5	2.9	3.3
100	50.5	5.8	6.6
1,000	500.5	9.0	9.97
10,000	5000.5	12.0	13.29

Table 9.1 Average Number of Comparisons

9-44

为什么需要多种计算方法？

9-45

案例研究：百度面试题

3）百度面试题（可选）。

有一个N*N的矩阵A。，N=2°。每行的数是从小到大排列，每列数也是从小到大排列。请你判断一个数x，是否存在矩阵A中。 例如，4*4矩阵A：

1	3	7	10
2	4	8	11
3	6	9	15
7	8	10	17

请用自然语言描述你的查找的算法思想。用5, 11, 15作为算法测试案例。

提示：1）请参照二分查找的思想；2）如何判断一个数是否在方阵内。

9-46

Stacks(栈)

- A **stack** is an abstract data type in which accesses are made at only one end
  - LIFO, which stands for Last In First Out
  - The insert is called ***Push*** and the delete is called ***Pop***
  - *Empty()* 检测栈中是否存在数据项

9-47

栈的应用

```
//input  n and base
Initialize stack S
WHILE n not Zero DO
   S.push(n mod base)
   n = n div base
END WHILE
WHILE not S.empty() DO
   k = S.pop()
   print k
END WHILE
```

请用一句话说明这个程序的功能

简单说明栈在这里的用途

9-48

Queues(队列)

- A **Queue** is an abstract data type in which items are entered at one end and removed from the other end
  - FIFO, for First In First Out
  - Like a waiting line in a bank or supermarket
  - No standard queue terminology
    - ***Enqueue, Enque, Enq, Enter,*** and ***Insert*** are used for the insertion operation
    - ***Dequeue, Deque, Deq, Delete,*** and ***Remove*** are used for the deletion operation.

9-49

游戏设计：贪吃蛇的游走算法

游戏文字描述：

假设一个10*10的字符矩阵看作蛇的生存空间，其中有一条长度5的蛇（HXXXX），“H”表示蛇头，“X”表示蛇身体。你可以使用“ADWS”分别控制蛇的前进方向“左右上下”，当蛇头碰到自己的身体或走出边界，游戏结束，否则蛇按你指定方向前进一步。

9-50

字符化的贪吃蛇游戏

初始10*10字符矩阵：

```
XXXXH-----
----------
----------
----------
----------
----------
----------
----------
----------
----------
```

游戏程序伪代码：

```
输出字符矩阵
WHILE not 游戏结束 DO
    ch＝等待输入
    CASE ch DO
      ‘A’:左前进一步，break
      ‘D’:右前进一步，break
      ‘W’:上前进一步，break
      ‘S’:下前进一步，break
    END CASE
    输出字符矩阵
END WHILE
输出 Game Over!!!
```

9-51

用队列表示蛇的位置

假设10*10字符矩阵list：

```
XXXXH-----
----------
----------
----------
----------
----------
----------
----------
----------
----------
```

snake(队列)：

4,0	←head
3,0	
2,0	
1,0	
0,0	←tail

snake向右移动：

?,?	←head
4,0	
3,0	
2,0	
1,0	←tail

9-52

作业

1. Bubble Sort the list: 33, 56, 17, 8, 95, 22。Make sure the final result is from small to large.
Write out the list after the 2nd pass. (10 points)

2. Give a sorted array as list={60,65,75,80,90,95}. Design an algorithm to insert the value of x into the sorted array. Then test the algorithm with value 50,67,99.
思考：为什么选择插入点在list头上、中间、尾巴上的三个数作为算法测试的数据，你能解释吗？

3. What is the state of the stack after the following sequence of Push and Pop operations?

Push “anne”; Push “get”; Push “your” ; Pop; Push “my” Push “gun”

9-53