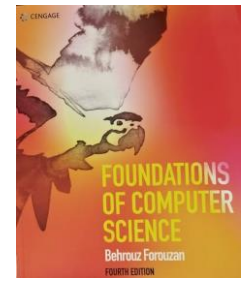
Chapter 12



Abstract Data Types

抽象資料型態

12.1 Source: Foundations of Computer Science © Cengage Learning

Objectives 學習目標

After studying this chapter, students should be able to:

- Define the concept of an abstract data type (ADT).
- Define a stack, the basic operations on stacks, their applications and how they can be implemented. 推疊定義及其運算
- Define a queue, the basic operations on queues, their applications and how they can be implemented.佇列定義及其運算
- Define a general linear list, the basic operations on lists, their applications and how they can be implemented.
- Define a general tree and its application.
- Define a binary tree—a special kind of tree—and its applications. 二元樹定義及其運算
- Define a binary search tree (BST) and its applications. 二元搜尋樹定義及其運算
- Define a graph and its applications.

12.2

ADTs 抽象資料型態

The definition of the data type and the definition of the operation to be applied to the data is part of the idea behind an **abstract data type (ADT)**定義資料型態及資料運算即為抽象資料型態的觀念— to hide how the operation is performed on the data. For example, the C language **defines a simple ADT as an integer, int**. **C語言已定義一些簡單的抽象資料型態，如整數 int.** The type of this ADT is an integer with predefined ranges. C also **defines several operations that can be applied to this data type** (**addition, subtraction, multiplication, division** and so on). **C語言也已定義一些基本資料運算，如加、減、乘、除等.** C explicitly defines these operations on integers and what we expect as the results.

12.3

Complex ADTs 複雜的抽象資料型態

Although **several simple ADTs**, such as **integer**, **real**, **character**, **pointer** and so on, have been implemented and are available for use in most languages, many useful complex ADTs are not. As we will see in this chapter, we need **a list ADT 串列的抽象資料型態, a stack ADT 推疊的抽象資料型態, a queue ADT 佇列的抽象資料型態** and so on. To be efficient, these ADTs should be created and stored in the library of the computer to be used.

12.4

STACKS 堆疊

A stack is a restricted linear list in which all additions and deletions are made at one end, the top. 推疊是一個有嚴格限制的線性串列，所有增加或刪除一筆資料都從頂端. If we insert a series of data items into a stack and then remove them, the order of the data is reversed. This reversing attribute is why stacks are known as **last in, first out** 資料後進先出**(LIFO) or first in, last out** 資料先進後出**(FILO) data structures.**

12.5

Operations on stacks 堆疊的運算

There are **four basic operations,** ***stack*****,** ***push*****,** ***pop*** **and** ***empty***, that we define in this chapter.

```
push (S,num)
{   top = top +1;
    S[top] = num;
}
```

```
pop (S)
{   num = S[top];
    top = top -1;
    return num;
}
```

12.6

Stack applications 堆疊之各種應用

Stack applications can be classified into four broad categories: ***reversing data反轉資料***, ***pairing data配對資料***, ***postponing data usage延後資料使用*** and ***backtracking steps反向追蹤***.

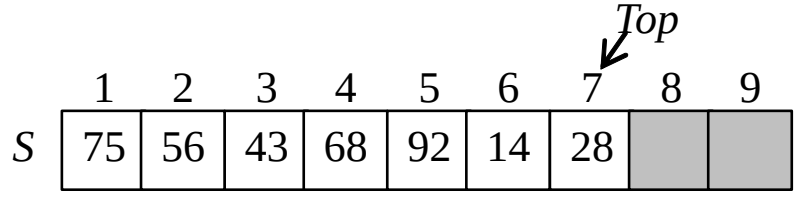
Stack implementation 推疊的實作

Stack ADTs can be implemented using either an **array** or a **linked list**. 推疊可使用陣列或鍊結列來實作.

In **array implementation**, we have a record that has two fields. The first field can be used to store information about the array. The **linked list implementation** is similar: we have an extra node that has the name of the stack. This node also has two fields: **a counter** and **a pointer** that points to the top element.

Exercise

Give a stack S as below,
determine Top =_____ and S[top]=_____ after Pop(S), Push(S,25), Push(S,33), Pop(S), Pop(S), and Pop(S).



QUEUES 佇列之抽象資料型態

A **queue** is a linear list in which data can only be inserted at one end, called the ***rear***, and deleted from the other end, called the ***front***. 佇列是一線性串列，插入一筆資料從尾端**rear**；刪除一筆資料從前端**front**. These restrictions ensure that the data is processed through the queue in the order in which it is received. In other words, a queue is a **first-in first-out (FIFO)資料先進先出 or last-in last-out (LILO)**或資料後進後出 structure.

12.9

Operations on queues 佇列的運算

Although we can define many operations for a queue, four are basic: ***queue***, ***enqueue***, ***dequeue*** and ***empty***, as defined below.

```
enqueue (Q,num)
{   rear = rear +1;
    if (front ==-1) front = rear;
    Q[rear] = num;
}
```

```
dequeue (Q)
{  num = Q[front];
   front = front +1;
   if (front ==-1) rear = front;
   return num;
}
```

12.10

Queue applications 佇列的應用

Queues are used in online business applications such as processing customer requests, jobs and orders. 佇列應用在線上電子商務如處理校費者的需求、業務與訂單等.

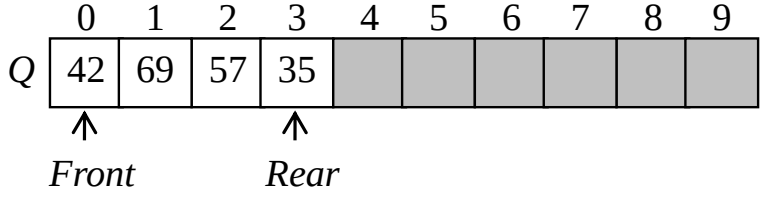
Queue implementation 佇列的實作

A queue ADT can be implemented using either **an array 陣列** or **a linked list 鏈結**. In the **array implementation** we have a record with three fields. The first field can be used to store information about the queue.The **linked list implementation** is similar: we have an extra node that has the name of the queue. This node also has three fields: a count, a pointer that points to the front element and a pointer that points to the rear element.

12.11

Exercise

Please show the contents of the queue *Q* after the operations of **enqueue(*Q*,54), dequeue(*Q*), enqueue(*Q*,71), dequeue(*Q*), dequeue(*Q*), and enqueue(*Q*,33), and *Front*=? and *Rear*=?**



12.12

TREES 樹

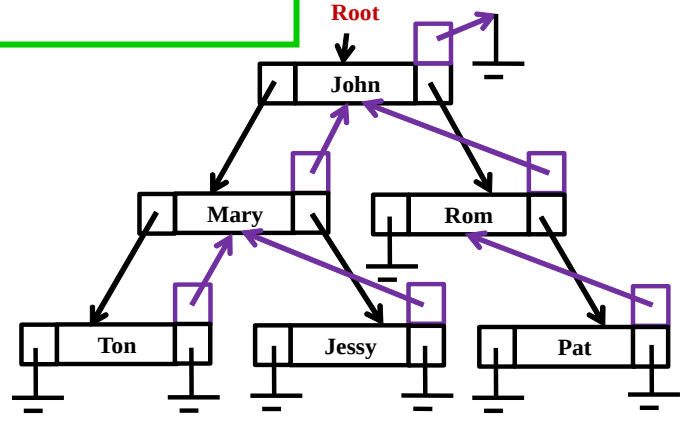
A **tree** consists of a finite set of elements, called **nodes** (or **vertices**) and a finite set of directed **lines**, called **arcs**, that connect pairs of the nodes. 樹由有限的節點與弧線所組成，含有樹根、中間節點及樹葉等

Each node in a tree may have a subtree. 樹的節點可以有子樹 The subtree of each node includes one of its children and all descendents of that child.每顆子樹可包含兒子與孫子

12.13

Physical structure of a tree, each node includes one of its children and all descendents of that child. 樹的實體結構，每個節點包含其兒子與孫子

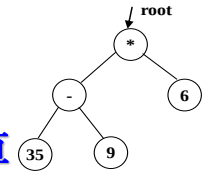
```
typedef struct node {
  char          name[20];
  struct node *lchild, *rchild, *father;
} NODE ;
```



12.14

BINARY TREES 二元樹

A binary tree is a tree in which no node can have more than two subtrees. 二元樹中每個節點最多兩顆子樹 In other words, a node can have zero, one or two subtrees.



Operations on binary trees 二元樹的運算

The **six most common operations** defined for a binary tree 二元樹運算有六種 are ***tree*建立空樹** (creates an empty tree), ***insert*插入一節點資料**, ***delete*刪除一節點資料**, ***retrieve*檢查一節點資料**, ***empty*樹是否為空的** and ***traversal*追蹤一顆樹所有節點資料**.

12.15

Binary tree traversals 二元樹之追蹤 (所有節點資料)

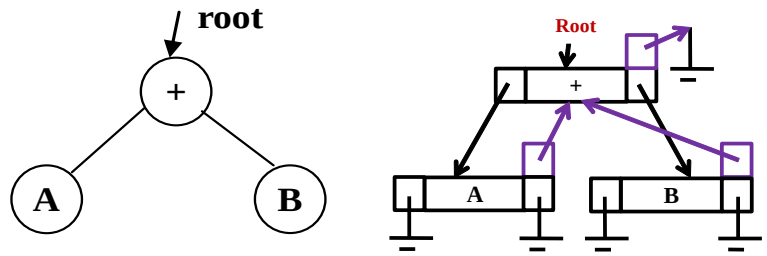
A binary tree traversal **requires that each node of the tree be processed once** and only once in a predetermined sequence.二元樹追蹤為在預訂的順序下，每個節點僅被拜訪一次 The two general approaches to the traversal sequence are兩種追蹤順序為 **depth-first深度優先** and **breadth-first廣度優先** traversal.

12.16

Expression trees 數學式之樹

An arithmetic expression can be represented in three different formats: 算術式能用三種來表示 **prefix前置式, infix中置式, and postfix後置式**.

Prefix: **+AB**
Infix: **A+B**
Postfix: **AB+**



12.17

BINARY SEARCH TREE 二元搜尋樹

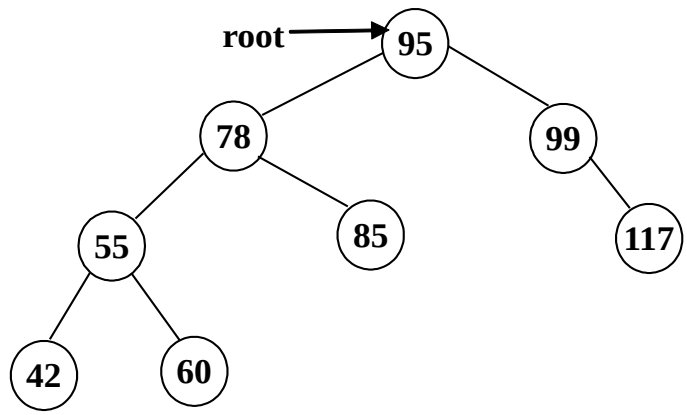
A **binary search tree** (**BST**) is a binary tree with one extra property: the key value of each node is greater than the key values of all nodes in each left subtree and smaller than the value of all nodes in each right subtree. 二元搜尋樹的定義: 小於該節點資料置於左子樹；大於該節點資料置於右子樹

12.18

Exercise: Please build a binary search tree for a series of numbers, 59, 64, 42, 85, 78, 36, 98, and 89. 請建立一顆二元搜尋樹 對此一連串數值 (根為59)

12.19

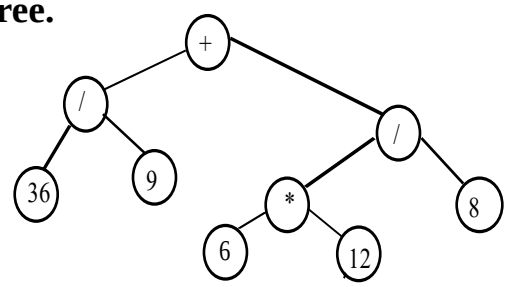
A very interesting property of a BST is that if we **apply the inorder traversal of a binary search tree, the elements that are visited are sorted in ascending order**.
對二元搜尋樹作中序追蹤之結果:
42, 55, 60,78, 85, 95, 99, 117 其結果即為由小而大的排序



12.20

Review Questions

- What are the **major differences** between **Stack** and **Queue ADTs**?
- Please define a **Stack** and **show their algorithms of push(S,data)** and **pop(S)**.
- Please define a **Queue** and **show their algorithms of enqueue(Q,data)** and **dequeue(Q)**.
- **Please show the preorder, inorder, and postorder for the binary tree.**



12.21