



**Object Oriented Software Development**

6. Arrays and collections extra notes:  
Implementing a Dictionary



---

---

---

---

---

---

---

---

**Dictionary ADT**

- A **dictionary** is a searchable collection of **key-item** pairs
- Also known as **map**, **associative list**
- Keys are used to locate items
- Collection can be sorted or unsorted
- May or may not allow more than one item with the same key – depends on implementation

GCU Logo Object Oriented Software Development 6. Arrays and collections extra notes 2

---

---

---

---

---

---

---

---

**Dictionary operations**

- **Put**(key, item) - add a new key/item pair
- **Get**(key) – get the item with the specified key
- **Set**(key, item) – set the item for the specified key
- **Remove**(key) – remove the item with the specified key
- **ContainsKey**(key) – check whether the specified key exists

GCU Logo Object Oriented Software Development 6. Arrays and collections extra notes 3

---

---

---


---

---

---

---

---



### Dictionary data types

- Most reusable case is when key and item data types are both Object
- Can store any type and use any type as key because of polymorphism
- Sometimes it is convenient to have a customised implementation restricted to specific data types
  - e.g. .NET Framework includes a StringDictionary class

GCU Object Oriented Software Development 6. Arrays and collections extra notes 4

---

---

---


---

---

---

---

---



### Implementing a dictionary

- Important property for a dictionary is fast lookup by key
- Need efficient search mechanism
- Linear search (looking through keys in turn until target is found) is very inefficient
- Two commonly used alternatives:
  - **Hashing**
  - **Binary search**

GCU Object Oriented Software Development 6. Arrays and collections extra notes 5

---

---

---


---

---

---

---

---



### Hashing

- A hash is a function that takes an item of whatever data type you are storing and outputs an integer in a certain range
- A good hash function is one that gets a fairly even distribution of the numbers in the output range
- Example: if the required range is 0-19, then the function **Value Mod 20** (remainder after dividing by 20) must give a result within the range

GCU Object Oriented Software Development 6. Arrays and collections extra notes 6

---

---

---


---

---

---

---

---



### Hash table

- A hash table is a dictionary implementation using hashing
- Two arrays or lists, one for keys, one for items
- Calculate hash of key – this determines at what position in the array the key is stored
- Item stored at same position in its array
- Ideally have entries spread out in arrays, in no particular order

GCU Object Oriented Software Development 6. Arrays and collections extra notes 7

---

---

---

---

---


---

---

---

---

---



### Hashing example

- Store objects of type Car with the registration number used as the key
- calculate hash code of registration number, to be within range 0-19
- Value is sum of Ascii codes of characters
- Hash code = Value Mod 20

<i>SD59DFF</i>	$28+13+5+9+13+15+15 = 98:$	<b>Hashcode = 18</b>
<i>FG10GTH</i>	$15+16+1+0+16+29+17 = 94:$	<b>Hashcode = 14</b>
<i>DR10GHY</i>	$13+27+1+0+16+17+34 = 108:$	<b>Hashcode = 8</b>
<i>FT56RET</i>	$15+29+5+6+27+14+29 = 125:$	<b>Hashcode = 5</b>
<i>NN59FRT</i>	$23+23+5+9+15+27+29 = 131:$	<b>Hashcode = 11</b>

GCU Object Oriented Software Development 6. Arrays and collections extra notes 8

---

---

---

---

---


---

---

---

---

---



### Hash table

POS	STATUS	KEYS	ITEMS
0:	empty		
1:	empty		
2:	empty		
3:	empty		
4:	empty		
5:	occupied	FT56RET	(Make: Honda, Reg: FT56RET)
6:	empty		
7:	empty		
8:	occupied	DR10GHY	(Make: Toyota, Reg: DR10GHY)
9:	empty		
10:	empty		
11:	occupied	NN59FRT	(Make: Jaguar, Reg: NN59FRT)
12:	empty		
13:	empty		
14:	occupied	FG10GTH	(Make: Ford, Reg: FG10GTH)
15:	empty		
16:	empty		
17:	empty		
18:	occupied	SD59DFG	(Make: Mazda, Reg: SD59DFG)
19:	empty		

GCU Object Oriented Software Development 6. Arrays and collections extra notes 9

---

---

---

---

---


---

---

---

---

---



### Searching by hashing

- Finding a hash table entry by key is **very fast**
- No need to search through items
- Process is as follows:
  - Calculate hash code of target key
  - This should give index of target in the hash table array
  - Go straight to that position and retrieve item

Object Oriented Software Development 6. Arrays and collections extra notes 10

---

---

---


---

---

---

---

---



### It's not quite that simple

- There always exists the possibility that two keys will hash to the same integer value
- When this happens, a **collision** results
- For example:
 

SD56QWS     $28+13+5+6+26+32+28 = 138$ ;    **Hashcode = 18** already occupied
- Need to implement techniques to deal with this situation
  - Closed hashing (linear probing, rehashing), open hashing

Object Oriented Software Development 6. Arrays and collections extra notes 11

---

---

---


---

---

---

---

---



### Dictionary example (hash table)

- **CollectionsDemo solution**
- **SimpleCollections project**
- **SimpleDictionary.cs**
- Implements a dictionary as a hash table
- .NET Framework has a Dictionary class which is essentially a hash table

Object Oriented Software Development 6. Arrays and collections extra notes 12

---

---

---

---

---

---

---

---

### Binary search

- Algorithm:
  - Compare the target to the middle item in the list
  - If the target is the same as the middle item, you've found the target
  - If it's before the middle item, repeat this procedure on the items before the middle
  - If it's after the middle item, repeat on the items after the middle
  - The method halves the number of items to check each time

---

---

---

---

---

---

---

---

### Binary search

- Requires that the elements in the collection are sorted
- The maximum number of comparisons required to find a target in an array of  $n$  elements is (the number of times that  $n$  can be divided in two) + 1.
- To search an array of 1024 elements would take **at most 10** comparisons using a binary search, but could take **up to 1024** comparisons using a linear search

---

---

---

---

---

---

---

---

### Binary search tree (BST)

- Can implement a Dictionary as a BST
- A hierarchical structure in which data access is similar to a binary search algorithm

```

    graph TD
      Jim[Jim] --> Dix[Dix]
      Jim --> Ron[Ron]
      Dix --> Amy[Amy]
      Dix --> Gay[Gay]
      Amy --> Ann[Ann]
      Gay --> Eve[Eve]
      Gay --> Jan[Jan]
      Ron --> Kay[Kay]
      Ron --> Tim[Tim]
      Kay --> Joan[Joan]
      Kay --> Kim[Kim]
      Tim --> Bob[Bob]
      Tim --> Tom[Tom]
    
```

---

---

---

---

---

---

---

---

### BST rules

- Consists of a node called the root, together with two children called the **left subtree** and the **right subtree** of the root
- Each of these children **is itself a binary tree**
- Each element has a key value which is used to order the elements
- The keys of all the elements in the left subtree of the root precede the key in the root
- The key in the root precedes all keys in the right subtree

Object Oriented Software Development 6. Arrays and collections extra notes 16

---

---

---

---

---

---

---

---

### BST nodes

- Each node contains data AND a references to the left and right subtrees
- An empty subtree is represented by a NULL reference
- Each subtree is itself a binary search tree

Object Oriented Software Development 6. Arrays and collections extra notes 17

---

---

---

---

---

---

---

---

### Dictionary example (BST)

- **CollectionsDemo solution**
- **SimpleCollections project**
- **SimpleBST.cs**
- Implements a basic dictionary as a BST
- Note that key must implement IComparable interface – e.g. String
- .NET Framework has a SortedDictionary class which is essentially a BST

Object Oriented Software Development 6. Arrays and collections extra notes 18

---

---

---

---

---

---

---

---

### Comparing hash table and BST



- Hash table is theoretically faster for searching
- Hash tables only find on equality searches, while trees can do range searches with things like  $<$ ,  $>$ ,  $<=$ ,  $>=$
- Trees allow in-order traversal



---

---

---

---

---

---

---

---