Data Structures

Shahed University – 2019

Ref: Ditle Visual C#

By: MH.Movasaghinia

Self-Referential Classes

- A **self-referential class** contains a reference member that refers to an object of the same classtype.

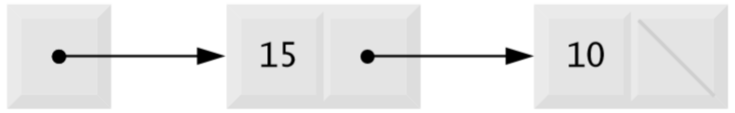


Fig. 19.2 | Self-referential class objects linked together.

Linked Lists

- A **linked list** is a linear collection (i.e., a sequence) of self-referential class objects, called nodes, connected by reference links—hence, the term "linked" list.

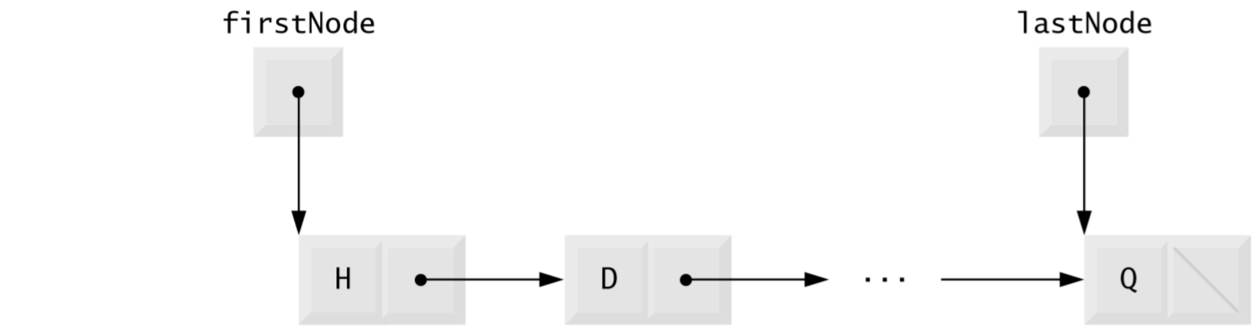


Fig. 19.3 | Linked list graphical representation.

Ref-Code : Fig. 19.4: LinkedListLibrary.cs

```
The list is: True

The list is: $ True

The list is: $ True 34567

The list is: $ True 34567 hello

$ removed
The list is: True 34567 hello

True removed
The list is: 34567 hello


hello removed
The list is: 34567

34567 removed
Empty list
```

Fig. 19.5 | Testing class `List`. (Part 3 of 3.)

Ref-Code : Fig. 19.5: ListTest.cs

Linear and Circular Singly Linked and Doubly Linked Lists

- **Circular ,singly linked list**

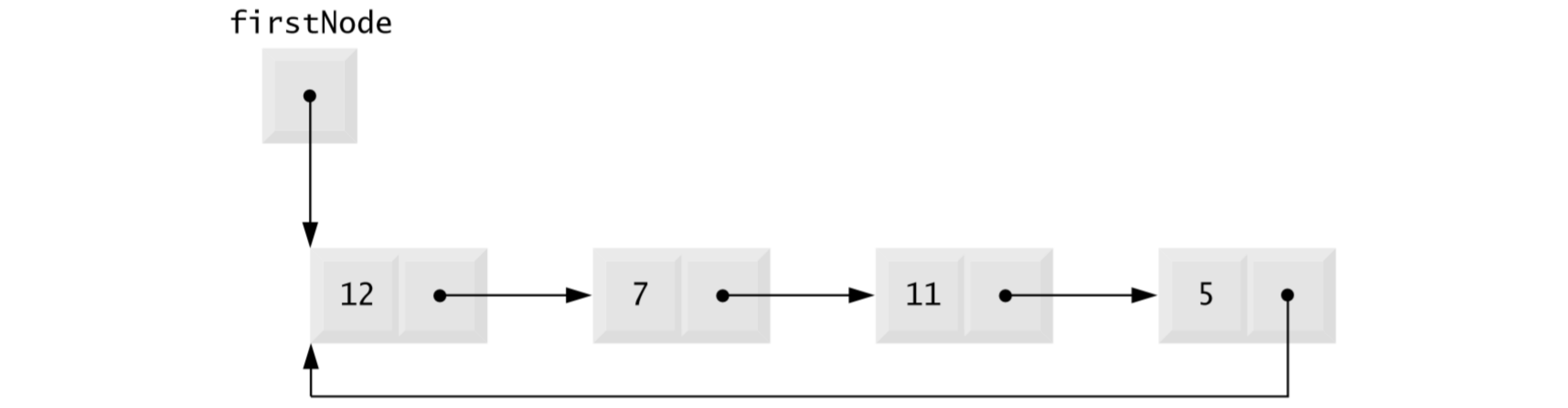


Fig. 19.10 | Circular, singly linked list.

Linear and Circular Singly Linked and Doubly Linked Lists

- **Doubly linked list**

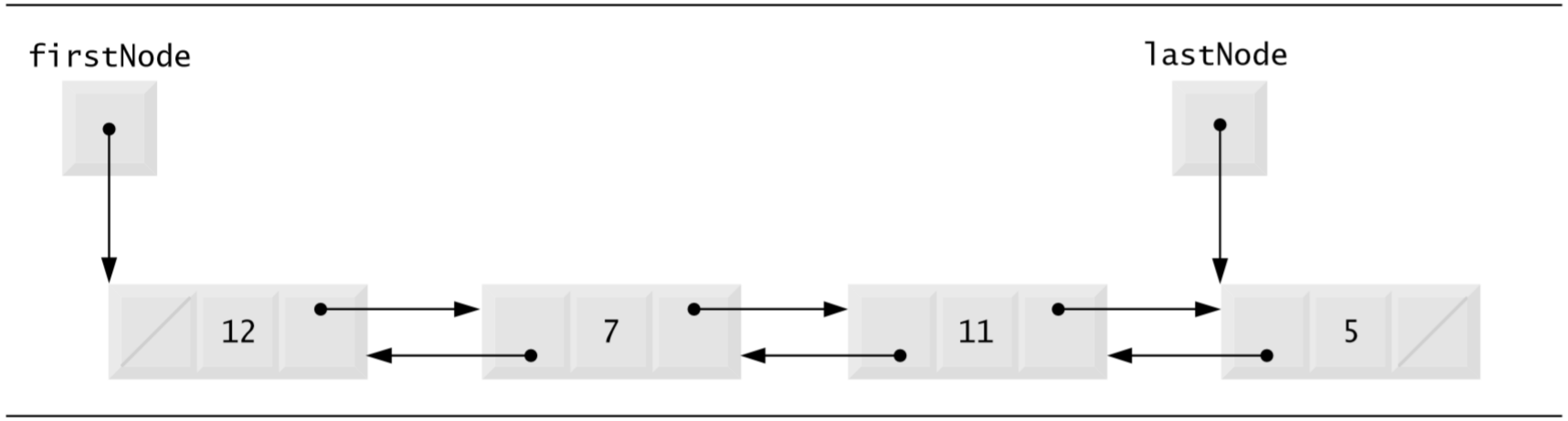


Fig. 19.11 | Doubly linked list.

Linear and Circular Singly Linked and Doubly Linked Lists

- **Circular, doubly linked list**

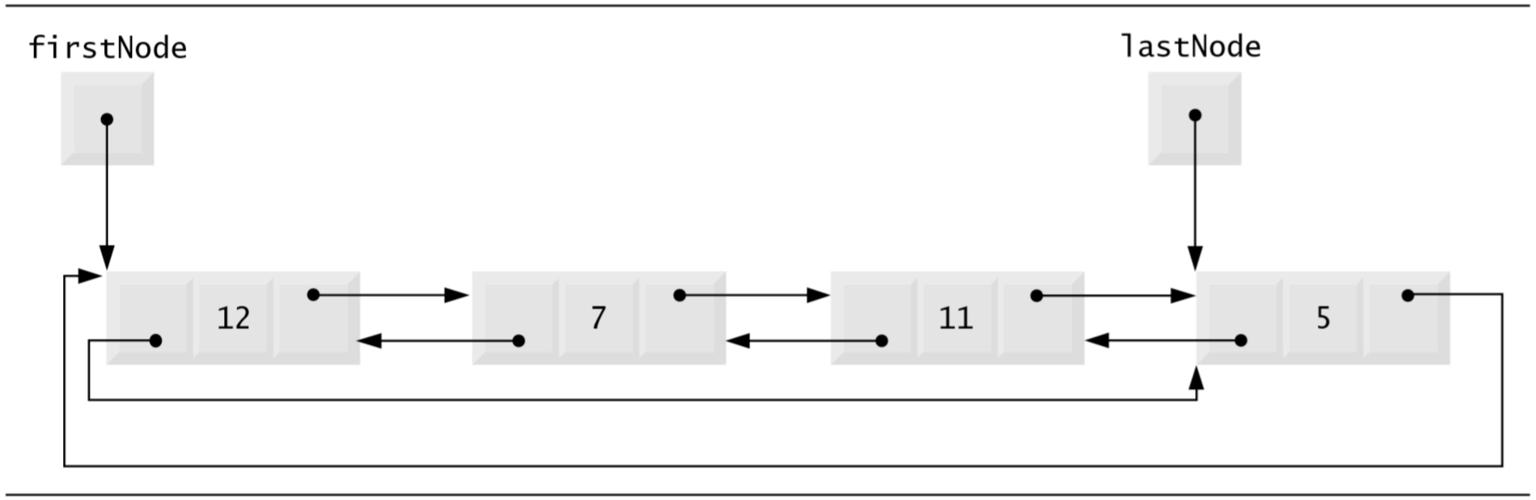
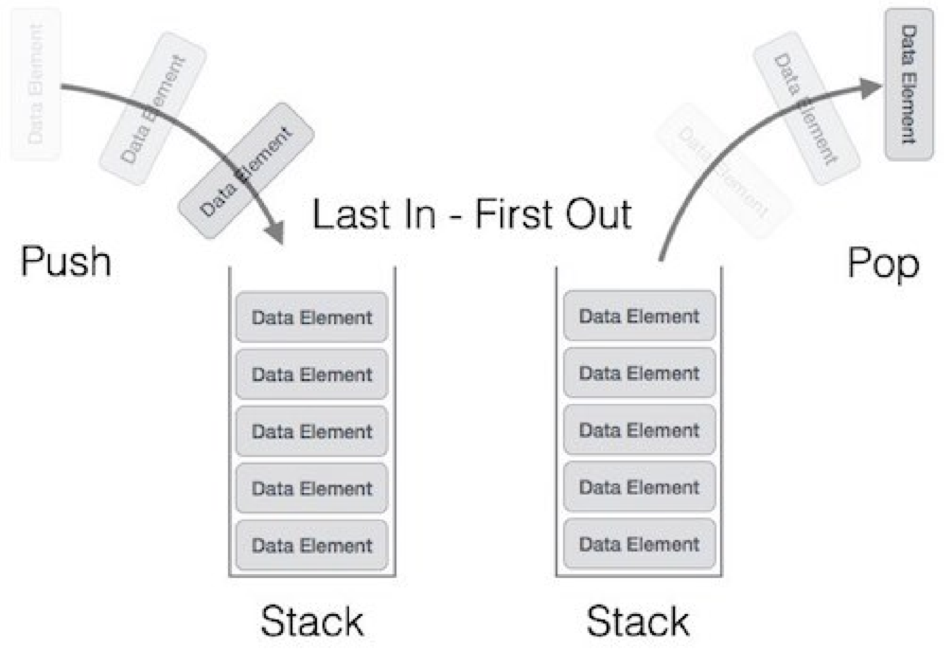


Fig. 19.12 | Circular, doubly linked list.

Stacks

- A stack may be viewed as a constrained version of a linked list—it receives new node sand releases nodes only at the top. For this reason, a stack is referred to as a last-in, first-out (LIFO) data structure.



Ref-Code : Fig. 19.13: StackInheritanceLibrary.cs

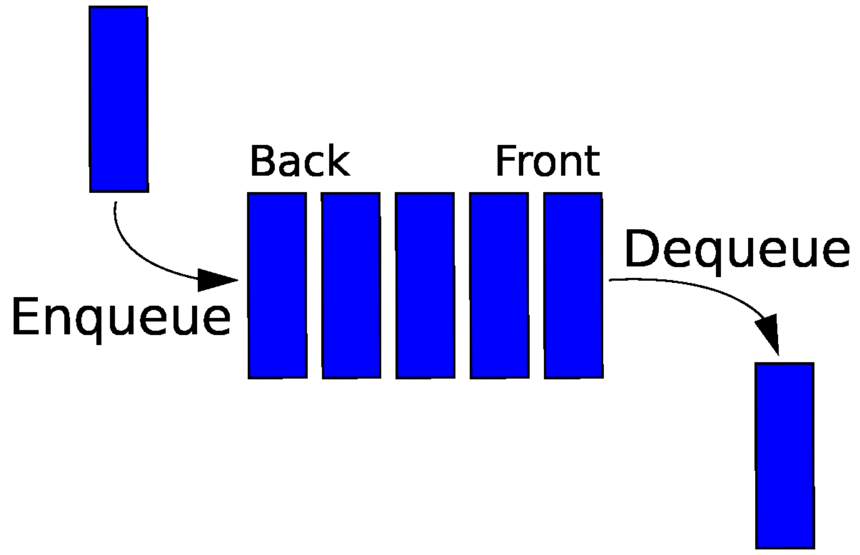
```
The stack is: True

The stack is: $ True

The stack is: 34567 $ True

The stack is: hello 34567 $ True

hello popped
The stack is: 34567 $ True

34567 popped
The stack is: $ True

$ popped
The stack is: True

True popped
Empty stack
   at LinkedListLibrary.List.RemoveFromFront()
      in C:\examples\ch21\Fig21_04\LinkedListLibrary\
      LinkedListLibrary\LinkedListLibrary.cs:line 78
   at StackInheritanceLibrary.StackInheritance.Pop()
      in C:\examples\ch21\Fig21_13\StackInheritanceLibrary\
      StackInheritanceLibrary\StackInheritance.cs:line 27
   at StackInheritanceTest.Main(String[] args)
      in C:\examples\ch21\Fig21_14\StackInheritanceTest\
      StackInheritanceTest\StackInheritanceTest.cs:line 35
```

Fig. 19.14 | Testing class StackInheritance. (Part 3 of 3.)

Ref-Code : Fig. 19.14: StackInheritanceTest.cs

Queues

- A queue is similar to a checkout line in a supermarket—the cashier services the person at the beginning of the line first. Other customers enter the line only at the end and wait for service. Queue nodes are removed only from the head (or front) of the queue and are inserted only at the tail (or end). For this reason, a queue is a first-in, first-out (FIFO) data structure. The insert and remove operations are known as enqueue and dequeue.



Ref-Code : Fig. 19.16: QueueInheritanceLibrary.cs

```
The queue is: True

The queue is: True $

The queue is: True $ 34567

The queue is: True $ 34567 hello

True dequeued
The queue is: $ 34567 hello

$ dequeued
The queue is: 34567 hello

34567 dequeued
The queue is: hello

hello dequeued
Empty queue
   at LinkedListLibrary.List.RemoveFromFront()
      in C:\examples\ch21\Fig21_04\LinkedListLibrary\
      LinkedListLibrary\LinkedListLibrary.cs:line 78
   at QueueInheritanceLibrary.QueueInheritance.Dequeue()
      in C:\examples\ch21\Fig21_16\QueueInheritanceLibrary\
      QueueInheritanceLibrary\QueueInheritance.cs:line 28
   at QueueTest.Main(String[] args)
      in C:\examples\ch21\Fig21_17\QueueTest\
      QueueTest\QueueTest.cs:line 38
```

Fig. 19.17 | Testing class `QueueInheritance`. (Part 2 of 2.)

Ref-Code : Fig. 19.17: QueueTest.cs

Trees

- Linked lists, stack sand queues are linear data structures (i.e., sequences).A tree is a non linear, two-dimensional data structure with special properties. Tree nodes contain two or more links.

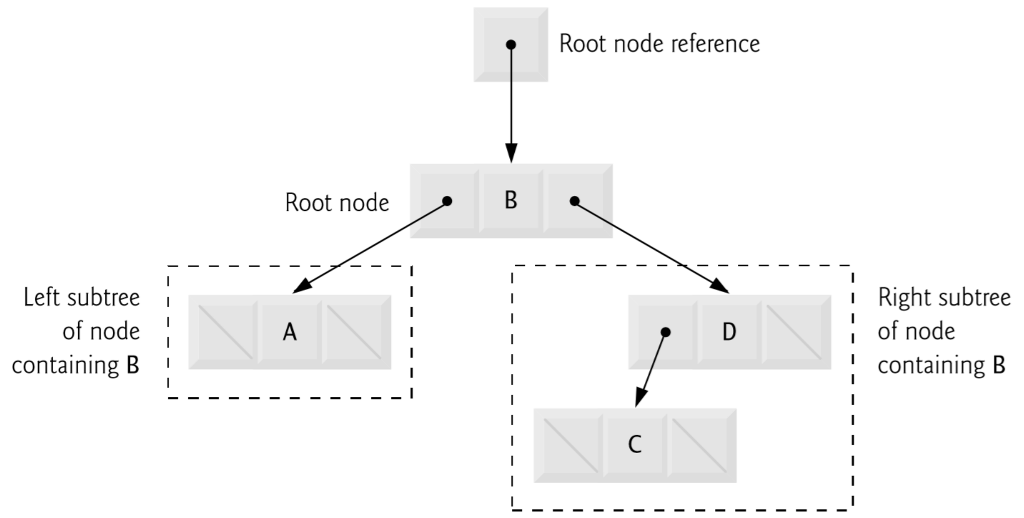

Root node reference
Root node
B
Left subtree of node containing B
A
Right subtree of node containing B
D
C

Fig. 19.18 | Binary-tree graphical representation.

Ref-Code : Fig. 19.20: BinaryTreeLibrary.cs

```
Inserting values:
39 69 94 47 50 72 55 41 97 73

Preorder traversal
39 69 47 41 50 55 94 72 73 97

Inorder traversal
39 41 47 50 55 69 72 73 94 97

Postorder traversal
41 55 50 47 73 72 97 94 69 39
```

Fig. 19.21 | Testing class `Tree` with a binary tree. (Part 2 of 2.)

Ref-Code : Fig. 19.21: TreeTest.cs

Some Helpful Applications

- https://github.com/boxgames1
- Show All Traverse of Binary Tree
  - https://github.com/zamhaq/react-binary-tree
  - https://github.com/boxgames1/binary-trees-app