**Student Name:** Federico Watkins

**CSIS 1800:** Introduction to Computer Science and Information Systems

**Chapter number:** 8 Abstract Data Types

**Assignment number:** 8

1. Array-based implementation and linked implementation:
2. Define these terms using your own words

Array-based: Refers to an implementation of a container in which the items are stored in an array. Hence each item is found by means of their index within the array.

Link: Refers to an implementation of a container where the items are stored together with information on where the next item can be found. This is done by use of a pointer within each node.

1. What do they have in common?

They have in common that they are both implementation of a list. And every list is characterized by the following three properties:

* The items are homogeneous
* The items are linear
* List have varying lengths

1. What distinguishes one from the other?

Already distinguished in part ‘a’ of this question. In an array-based implementation we see a fix structure in which the *length* must be explicitly kept. In a linked implementation we see an unbounded list in which the nodes are created at run time and whose only limit on the number of nodes is the size of memory.

(Computer Science Illuminated, Nell Dale and John Lewis, 4th Edition, p. 246-251)

1. Draw the unsorted list containing the following strings: *black, purple, white, yellow, red, blue, violet, and green.*
2. In an unsorted array-based list

|  |  |  |
| --- | --- | --- |
| **Length** | **List** | **Index** |
| 8 | Black | [0] |
|  | Purple | [1] |
|  | White | [2] |
|  | Yellow | [3] |
|  | Red | [4] |
|  | Blue | [5] |
|  | Violet | [6] |
|  | Green | [7] |
|  |  | [max\_length-1] |

1. In a sorted array-based list

|  |  |  |
| --- | --- | --- |
| **Length** | **List** | **Index** |
| 8 | Black | [0] |
|  | Blue | [1] |
|  | Green | [2] |
|  | Purple | [3] |
|  | Red | [4] |
|  | Violet | [5] |
|  | White | [6] |
|  | Yellow | [7] |
|  |  | [max\_length-1] |

1. In an unsorted linked list

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| List |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| Black |  |  | Purple |  |  | White |  |  | Yellow |  |
|  |  |  |  |  |  |  |  |  |  |  |
| Red |  |  | Blue |  |  | Violet |  |  | Green |  |

1. In a sorted linked list

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| List |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| Black |  |  | Blue |  |  | Green |  |  | Purple |  |
|  |  |  |  |  |  |  |  |  |  |  |
| Red |  |  | Violet |  |  | White |  |  | Yellow |  |

(Computer Science Illuminated, Nell Dale and John Lewis, 4th Edition, p. 246-251)

1. Define stacks and queues structures using your own words and give one example of each structure.

Stacks: An abstract data type in which access is only allowed through one end. An item may be inserted as the first one and only the first one may be removed. It is known as Last In First Out (LIFO)

Example: The plate holder at a cafeteria as we can only take the top plate and replace a plate to it on the top of the stack of plates. Canned goods on a grocer’s shelf as we take the first can in a row we take the last can put in that row.

Queue: An abstract data type in which items are entered at one end and removed from the other end. Insertions are made at the rear of the queue and deletions are made from the front of the queue. It is known as First In First Out (FIFO).

Example: A waiting line in a bank or supermarket where the last in would be the last out and vice-versa.

(Computer Science Illuminated, Nell Dale and John Lewis, 4th Edition, p. 246-251)

1. What is the content of the stack at each step? The values on the right side are static variables holding some values, while item is a temporary variable holding the last POP’ed element. In order to avoid any mistakes, write values vertically to demonstrate each step. Remember, First In, Last Out arrangement.

Push A

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
|  | [3] |
|  | [2] |
|  | [1] |
| A | [0] |

Push B

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
|  | [3] |
|  | [2] |
| B | [1] |
| A | [0] |

Push C

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
|  | [3] |
|  | [2] |
|  | [1] |
| A | [0] |

Pop item

|  |  |  |
| --- | --- | --- |
| **Stack content** | **Index** | **Pop’ed Item** |
|  | […] | C |
|  | [4] |  |
|  | [3] |  |
|  | [2] |  |
| B | [1] |  |
| A | [0] |  |

Push D

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
|  | [3] |
| D | [2] |
| B | [1] |
| A | [0] |

Push E

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
| E | [3] |
| D | [2] |
| B | [1] |
| A | [0] |

Push item

|  |  |  |
| --- | --- | --- |
| **Stack content** | **Index** | **Pop’ed Item** |
|  | […] | E |
|  | [4] |  |
|  | [3] |  |
| D | [2] |  |
| B | [1] |  |
| A | [0] |  |

(Computer Science Illuminated, Nell Dale and John Lewis, 4th Edition, p. 246-251)

1. What is the content of the queue at each step? In order to avoid any mistakes, write values horizontally to demonstrate each step. Remember, First In, First Out arrangement. Item is a temporary variable holding the last POP’ed element

Enqueue A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Head |  | A |  |  | Tail |

Enqueue B

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | A |  |  | B |  |  | Tail |

Enqueue C

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | A |  |  | B |  |  | C |  |  | Tail |

Dequeue item

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | B |  |  | C |  |  | Tail |  | **Dequeued** : A |

Enqueue D

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | B |  |  | C |  |  | D |  |  | Tail |

Enqueue E

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | B |  |  | C |  |  | D |  |  | E |  |  | Tail |

Dequeue item

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | C |  |  | D |  |  | E |  |  | Tail |  | **Dequeued** : B |

(Computer Science Illuminated, Nell Dale and John Lewis, 4th Edition, p. 246-251)

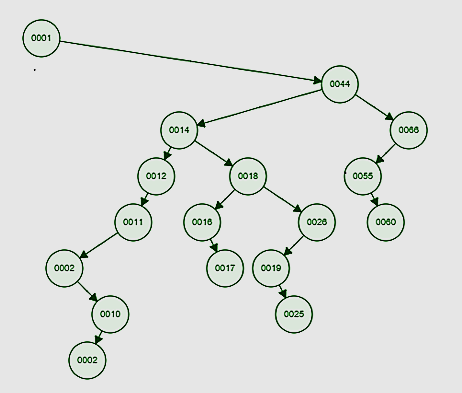
1. What are the properties of a binary tree and of a binary search tree?

Binary Tree: A container object in which each node is capable of having **two** successor nodes, called **children**. Each of the children, being nodes in the binary tree, can also have up to two child nodes, and these children can also have up to two children, and so on. The beginning of the tree is a unique starting node called the **root**, which is not the child of any node. A **unique path** exists from the root to every other node. If they exist, the node to the left of a parent node is called a **left child** and the other a **right child**. If a node in the tree has no children it is called a **leaf node**. Thus every node except the root has a unique parent. Nodes have levels which refer to their distance from the root.

(Computer Science Illuminated, Nell Dale and John Lewis, 4th Edition, p. 251-260)

1. Draw the binary search tree having the following elements:

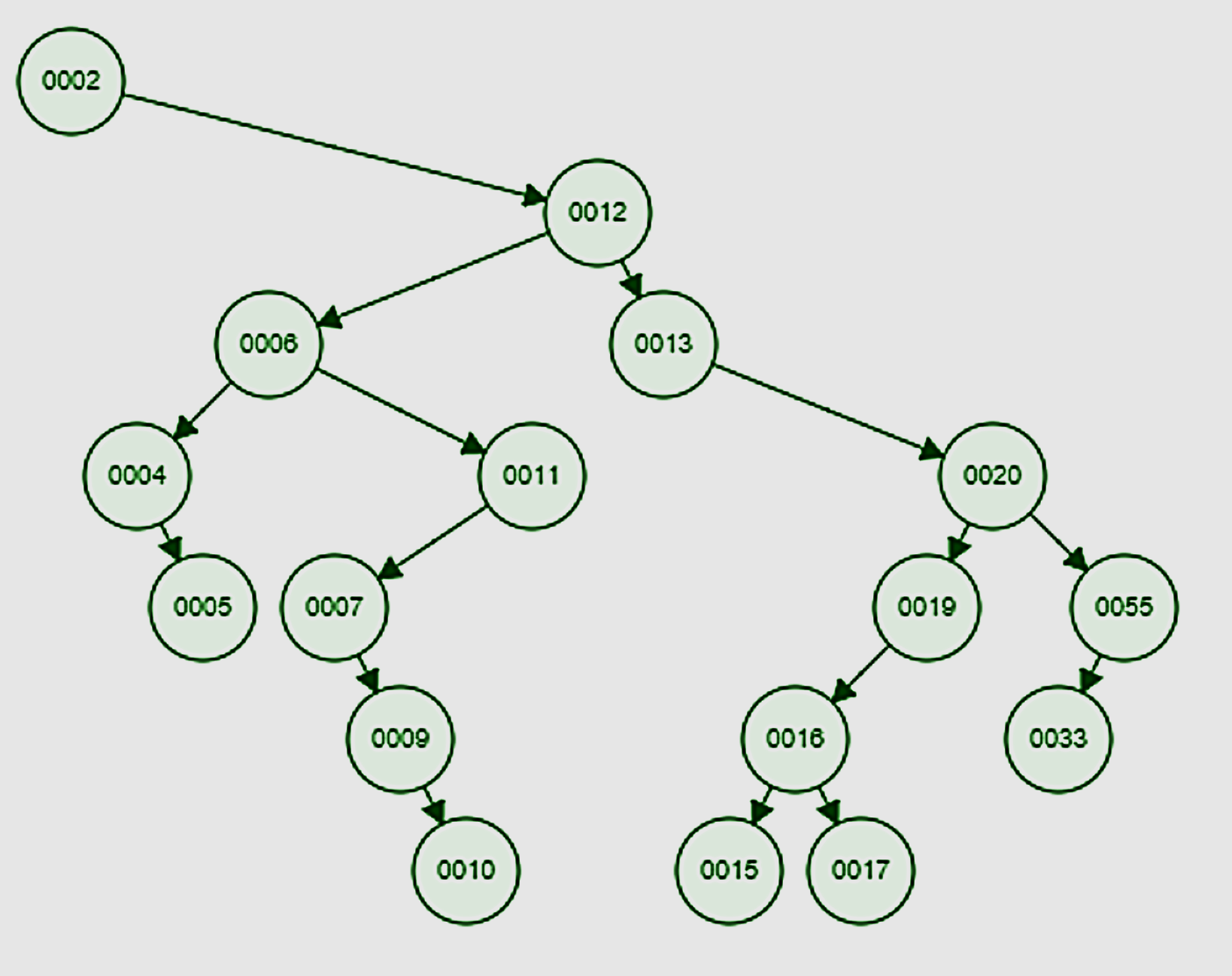
1, 44 14 66 55 18 26 12 11 19 2 10 25 2 16 17 60



(Computer Science Illuminated, Nell Dale and John Lewis, 4th Edition, p. 251-260)

1. Draw the binary search tree having the following elements:

2 12 6 4 11 7 13 9 20 10 55 19 16 17 5 33 15



(Computer Science Illuminated, Nell Dale and John Lewis, 4th Edition, p. 251-260)

1. What is the content of the stack at each step and values of item1 and item2? In order to avoid any mistakes, write values vertically to demonstrate each step. Remember, First In, Last Out arrangement. Item1, item2 are temporary variable holding the last POP’ed element

Push A

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
|  | [3] |
|  | [2] |
|  | [1] |
| A | [0] |

Push B

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
|  | [3] |
|  | [2] |
| B | [1] |
| A | [0] |

Pop item1

|  |  |  |
| --- | --- | --- |
| **Stack content** | **Index** | **Pop’ed Item** |
|  | […] | B |
|  | [4] |  |
|  | [3] |  |
|  | [2] |  |
|  | [1] |  |
| A | [0] |  |

Push C

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
|  | [3] |
|  | [2] |
| C | [1] |
| A | [0] |

Push D

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
|  | [3] |
| D | [2] |
| C | [1] |
| A | [0] |

Pop item2

|  |  |  |
| --- | --- | --- |
| **Stack content** | **Index** | **Pop’ed Item** |
|  | […] | D |
|  | [4] |  |
|  | [3] |  |
|  | [2] |  |
| C | [1] |  |
| A | [0] |  |

Push item1

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
|  | [3] |
| B | [2] |
| C | [1] |
| A | [0] |

Push E

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
|  | [4] |
| E | [3] |
| B | [2] |
| C | [1] |
| A | [0] |

Push item2

|  |  |
| --- | --- |
| **Stack content** | **Index** |
|  | […] |
| D | [4] |
| E | [3] |
| B | [2] |
| C | [1] |
| A | [0] |

(Computer Science Illuminated, Nell Dale and John Lewis, 4th Edition, p. 246-251)

1. What is the content of the queue at each step and values of item1 and item2? In order to avoid any mistakes, write values horizontally to demonstrate each step. Remember, First In, First Out arrangement. Item1, item2 are temporary variable holding the last POP’ed element

Enqueue A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Head |  | A |  |  | Tail |

Enqueue B

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | A |  |  | B |  |  | Tail |

Enqueue C

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | A |  |  | B |  |  | C |  |  | Tail |

Dequeue item1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | B |  |  | C |  |  | Tail |  | **Dequeued** : A |

Enqueue D

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | B |  |  | C |  |  | D |  |  | Tail |

Enqueue E

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | B |  |  | C |  |  | D |  |  | E |  |  | Tail |

Dequeue item2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | C |  |  | D |  |  | E |  |  | Tail |  | **Dequeued** : B |

Enqueue F

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | C |  |  | D |  |  | E |  |  | F |  |  | Tail |

Enqueue item1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  | C |  |  | D |  |  | E |  |  | F |  |  | A |  |  | Tail |

(Computer Science Illuminated, Nell Dale and John Lewis, 4th Edition, p. 246-251)

Additional References:

1. <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Stacks%20and%20Queues/Stacks%20and%20Queues.html>
2. <http://www.studytonight.com/data-structures/queue-data-structure>
3. <http://algs4.cs.princeton.edu/32bst/>