Answers are in blue.

# For Exercises 1–10, indicate which structure would be a more suitable choice for each of the following applications by marking them as follows:



**Computer Science Illuminated, Seventh Edition**

Nell Dale, PhD; John Lewis, PhD

**CHAPTER 8**

EXERCISES AND ANSWERS

## Stack

1. **Queue**

## Tree

1. **Binary search tree**

## Graph

* 1. A bank simulation of its teller operation to see how waiting times would be affected by adding another teller.

B

* 1. A program to receive data that is to be saved and processed in the reverse order.

A

* 1. An electronic address book ordered by name. D
  2. A word processor to have a PF key that causes the preceding command to be redisplayed. Every time the PF key is pressed, the program is to show the command that preceded the one currently displayed.

A

* 1. A dictionary of words used by a spelling checker to be built and maintained.

D

* 1. A program to keep track of patients as they check into a medical clinic, assigning patients to doctors on a first-come, first-served basis.

B

* 1. A program keeping track of where canned goods are located on a shelf.

A

* 1. A program to keep track of the soccer teams in a city tournament.

C

* 1. A program to keep track of family relationships. C or E
  2. A program to maintain the routes in an airline. E

# For Exercises 11–30, mark the answers true or false as follows:

## True

1. **False**
   1. A binary search cannot be applied to a tree. B
   2. A stack and a queue are different names for the same ADT. B
   3. A stack displays FIFO behavior. B
   4. A queue displays LIFO behavior. B
   5. A leaf in a tree is a node with no children. A
   6. A binary tree is a tree in which each node can have zero, one, or two children.

A

* 1. A binary search tree is another name for a binary tree. B
  2. The value in the right child of a node (if it exists) in a binary search tree will be greater than the value in the node itself. A
  3. The value in the left child of a node (if it exists) in a binary search tree will be greater than the value in the node itself. B
  4. In a graph, the vertices represent the items being modeled. A
  5. Algorithms that use a list must know whether the list is array based or linked.

B

* 1. A list may be linear or nonlinear, depending on its implementation.

B

* 1. The root of a tree is the node that has no ancestors. A
  2. Binary search trees are ordered. A
  3. On average, searching in a binary search tree is faster than searching in a list.

B

* 1. On average, searching in a binary search tree is faster than searching in a linked list.

A

* 1. A binary search tree is always balanced. B
  2. Given the number of nodes and the number of levels in a binary search tree, you can determine the relative efficiency of a search in the tree.

A

* 1. Insertion in a binary search tree is always into a leaf node. B
  2. A binary search tree is another implementation of a sorted list.

A

**The following algorithm (used for Exercises 31–33)** is a count-controlled loop going from 1 through 5. At each iteration, the loop counter is either printed or put on a stack, depending on the result of Boolean function RanFun(). (The behavior of RanFun() is immaterial.) At the end of the loop, the items on the stack are popped and printed. Because of the logical properties of a stack, this algorithm cannot print certain sequences of the values of the loop counter. You are given an output and asked if the algorithm could generate the output. **Respond as follows:**

## True

1. **False**

## Not enough information

Set count to 0 WHILE (count  5)

Set count to count  1 IF (RanFun())

Write count, ' ' ELSE

Push(myStack, count) WHILE (NOT IsEmpty(myStack))

Pop(myStack, number) Write number, ' '

* 1. The following output is possible using a stack: 1 3 5 2 4. B
  2. The following output is possible using a stack: 1 3 5 4 2. A
  3. The following output is possible using a stack: 1 3 5 1 3. B

**The following algorithm (used for Exercises 34–36)** is a count-controlled loop going from 1 through 5. At each iteration, the loop counter is either printed or put on a queue depend- ing on the result of Boolean function RanFun(). (The behavior of RanFun() is immaterial.) At the end of the loop, the items on the queue are dequeued and printed. Because of the logical proper- ties of a queue, this algorithm cannot print certain sequences of the values of the loop counter. You are given an output and asked if the algorithm could generate the output. **Respond as follows:**

## True

* + 1. **False**

## Not enough information

Set count to 0 WHILE (count  5)

Set count to count  1 IF (RanFun())

Write count, ' ' ELSE

Enqueue(myQueue, count) WHILE (NOT IsEmpty(myQueue))

Dequeue(myQueue, number) Write number, ' '

* 1. The following output is possible using a queue: 1 3 5 2 4. A
  2. The following output is possible using a queue: 1 3 5 4 2. B
  3. The following output is possible using a queue: 1 3 5 1 3. B

# Exercises 37–50 are short-answer questions.

* 1. What is written by the following algorithm?

Push(myStack, 5)

Push(myStack, 4)

Push(myStack, 4) Pop(myStack, item) Pop(myStack, item) Push(myStack, item)

WHILE (NOT IsEmtpy(myStack)) Pop(myStack, item)

Write item, ' '

4 5

* 1. What is written by the following algorithm?

Enqueue(myQueue, 5)

Enqueue(myQueue, 4)

Enqueue(myQueue, 4) Dequeue(myQueue, item) Dequeue(myQueue, item) Enqueue(myQueue, item)

WHILE (NOT IsEmtpy(myQueue)) Dequeue(myQueue, item) Write item, ' '

4 4

* 1. Write an algorithm that sets bottom equal to the last element in the stack, leaving the stack empty.

WHILE (NOT IsEmpty(myStack)) Pop(myStack, bottom)

* 1. Write an algorithm that sets bottom equal to the last element in the stack, leaving the stack unchanged.

WHILE (NOT IsEmpty(myStack)) Pop(myStack, bottom) push(newStack, bottom)

WHILE (NOT IsEmtpy(newStack)) Pop(newStack, item) Push(myStack, item)

* 1. Write an algorithm to create a copy of myStack, leaving myStack unchanged.

WHILE (NOT IsEmpty(myStack)) Pop(myStack, item) push(newStack, item)

WHILE (NOT IsEmtpy(newStack)) Pop(newStack, item) Push(myStack, item) Push(copyStack, item)

* 1. Write an algorithm that sets last equal to the last element in a queue, leaving the queue empty.

WHILE (NOT IsEmpty(myQueue)) Dequeue(myQueue, last)

* 1. Write an algorithm that sets last equal to the last element in a queue, leaving the queue unchanged.

WHILE (NOT IsEmpty(myQueue)) Dequeue(myQueue, last) Enqueue(newQueue, last)

WHILE (NOT IsEmtpy(newQueue)) Dequeue(newQueue, item) Enqueue(myQueue, item)

* 1. Write an algorithm to create a copy of myQueue, leaving myQueue unchanged.

WHILE (NOT IsEmpty(myQueue)) Dequeue(myQueue, last) Enqueue(newQueue, last)

WHILE (NOT IsEmtpy(newQueue)) Dequeue(newQueue, item) Enqueue(myQueue, item) Enqueue(copyQueue, item)

* 1. Write an algorithm Replace that takes a stack and two items. If the first item is in the stack, replace it with the second item, leaving the rest of the stack unchanged.

Replace(myStack, first, second) Pop(myStack, item)

WHILE (Item NOT equal furst) push(newStack, item) Pop(myStack, item)

IF (NOT IsEmpty(myStak)) Push(newStack, second)

WHILE (NOT IsEmtpy(newStack)) Pop(newStack, item) Push(myStack, item)

* 1. Write an algorithm Replace that takes a queue and two item. If the first item is in the queue, replace it with the second item, leaving the rest of the queue unchanged. Replace(myQueue, first, second)

Dequeue(myQueue, item) WHILE (Item NOT equal furst)

Enqueue(newQueue, item) Dequeue(myQueue, item)

IF (NOT IsEmpty(myQueue)) Enqueue(newQueue, second)

WHILE (NOT IsEmtpy(newQueue)) Dequeue(newQueue, item) Enqueue(myQueue, item)

* 1. Draw the binary search tree whose elements are inserted in the following order:

50 72 96 94 107 26 12 11 9 2 10 25 51 16 17 95

Insert tree like one on page 690 (11) of C++ Plus, 3rd edition. remove box and arrow as answer.

* 1. If Print is applied to the tree formed in Exercise 47, in which order would the elements be printed?

2 9 10 11 12 16 17 25 26 50 51 72 94 95 96 107

* 1. Examine the following algorithm and apply it to the tree formed in Exercise 47. In which order would the elements be printed?

Print2 (tree)

IF (tree is NOT null)

Print (right(tree)) // Recursive call R1 Write info(tree)

Print(left(tree)) // Recursive call R2

107 96 95 94 72 51 50 26 25 17 16 12 11 10 9 2

* 1. Examine the following algorithm and apply it to the tree formed in Exercise 47. In what order would the items be printed?

Print3 (tree)

IF (tree is NOT null)

Print (right(tree)) // Recursive call R1 Print(left(tree)) // Recursive call R2 Write info(tree)

107 95 94 96 51 72 17 16 25 10 2 9 11 12 26 50

# Exercises 51–55 are short answer questions based on the following directed graph.

*Alaska*

*Oregon*

*Vermont*

*Hawaii*

*New York*

*Texas*

* 1. Is there a path from Oregon to any other state in the graph? No
  2. Is there a path from Hawaii to every other state in the graph? Yes

*California*

* 1. From which state(s) in the graph is there a path to Hawaii? Texas
  2. Show the table that represents this graph.

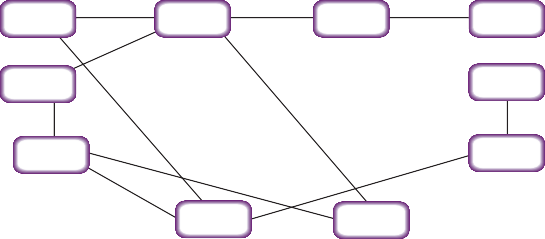
An F means that there in not a link; a T means there is a link.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Alaska | F | F | F | F | T | F | F |
| California | F | F | F | F | F | F | F |
| Hawaii | T | T | F | T | F | T | F |
| New York | F | F | F | F | F | F | F |
| Oregon | F | F | F | F | F | F | F |
| Texas | F | F | T | F | F | F | T |
| Vermont | T | T | F | T | F | F | F |
|  | Alaska | Calif. | Hawaii | New York | Oregon | Texas | Vermont |

* 1. Can you get from Vermont to Hawaii? No

# Exercises 56–60 are short answer questions based on the following directed graph.

EmployeeGraph



Jean

Susan

Darlene

Mike

John

Fred

Fran

Brent

Lance

Sandler

* 1. Show the depth-first traversal from Jean to Sander.

Jean, Lance, Fran, John, Susan, Sander (always taking the left- most arc if there is more than one)

* 1. Show the depth-first traversal from Lance to Darlene.

Lance, Fran, John, Susan, Darlene

* 1. Show the breadth-first traversal from Jean to Sander.

Jean, Lance, Susan, Fran, Brent, Sander

* 1. Show the breadth-first traversal from Lance to Darlene.

Lance, Fran, Jean, Brent, John, Susan, Darlene

* 1. Show the table that represents this graph.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Brent | F | F | F | T | F | F | T | F | F | F |
| Darlene | F | F | F | F | F | F | F | T | F | T |
| Fran | F | F | F | F | F | T | T | F | T | F |
| Fred | T | F | F | F | F | F | F | F | F | F |
| Jean | F | F | F | F | F | F | T | F | F | T |
| John | F | F | T | F | F | F | F | F | F | T |
| Lance | T | F | T | F | T | F | F | F | F | F |
| Mike | F | T | F | F | F | F | F | F | F | F |
| Sander | F | F | T | F | F | F | F | F | F | T |
| Susan | F | T | F | F | T | T | F | F | T | F |
|  | Brent | Darlene | Fran | Fred | Jean | John | Lance | Mike | Sander | Susan |

# Exercise 61–69 are short answer exercises.

* 1. Given the record List containing the array values and the vari- able length, write the algorithm for GetLength.

GetLength RETURN length

* 1. Assume that record List has an additional variable currentPo- sition, initialized to the first item in the list. What is the value of currentPosition?

0

* 1. Write the algorithm for MoreItems, which returns TRUE if there are more items in the list, and FALSE otherwise. MoreItems(list)

RETRUN List.currentPosition NOT equal to length

* 1. Write the algorithm for GetNext(myList, item) so that item is the next item in the list. Be sure to update currentPosition. GetNext(list, item)

Set item to list.values[list.currentPosition]

Set list.currentPosition to list.currentPosition  1

* 1. Exercises 61–64 create the algorithms that allow the user of a list to see the items one at a time. Write the algorithm that uses these operations to print the items in a list.

WHILE (MoreItems(list)) GetNext(list,item) Print(item)

* 1. What happens if an insertion or deletion occurs in the middle of an iteration through the list? Explain.

You cannot predict exactly what might happen, but you can be sure it probably will cause problems. It depends on where you are in the traversal. If you add an item at the end of an unsorted list, everything will probably be ok. If you add an item in a sorted list, you may miss the new item in the traversal. If you delete an item, it might have already processed the item.

* 1. Can you think of a way to keep the user from doing an inser- tion or deletion during an iteration?

As a part of the documentation, write that no operations that change the list can be made during an iteration. Such state- ments are called pre-conditions or assumptions.

* 1. Distinguish between value and reference parameters.

A value parameter is one for which a copy of the argu- ment is given to the subprogram. A reference parameter is one for which the address of the argument is given to the subprogram.

* 1. How are arguments and parameters matched?

Arguments and parameters are matched by position on the subprogram heading and the calling statement.