PES UNIVERSITY

PES University, Bengaluru
(Established under Karnataka Act No. 16 of 2013)

UE17CS207

NOV. 2018: LABORATORY B.TECH. IV SEMESTER

DATA STRUCTURES LAB (UE17CS207)

List of Lab Programs for Conduction:

Lab Program 01: Implement a singly linked list with insert and other operations

Lab Program 02: Implement a singly linked list with delete and other operations

Lab Program 03: Implement a doubly linked list with insert and other operations

Lab Program 04: Implement a doubly linked list with delete and other operations

Lab Program 05: Implement a stack using a singly linked list

Lab Program 06: Parentheses matching using stack data structure

Lab Program 07: Infix to Postfix conversion

Lab Program 08: Implement a queue using a singly linked list

Lab Program 09: Implement a circular queue using an array

Lab Program 10: Circular deadly game

Lab Program 11: Implement a queue using two stacks

Lab Program 12: Implement a BST and print the tree traversals

Lab Program 13: Implement a BST and find the leaf and non-leaf count

Lab Program 14: Construction of max-heap

Lab Program 15: Implement Priority Queue using min-heap

Lab Program 01: Implement a singly linked list with insert and other operations

The following operations are to be implemented in a Singly Linked List:
a) Insert an element at a specified position
b) Delete the element at the end of the list
c) Reverse the nodes in the list
d) Display the elements of the list

Input Format: Every new line has one of the following operation code and any data needed for the operation with a ‘space’ in between. The program terminates on termination operation.

0 - Insert an element at the specified position. In addition to the operation code, an element value and the offset after which the element to be inserted are provided. The element to be inserted after "offset" number of nodes. An offset value of '0' indicates the beginning of the list and an offset value of the length of the list indicates the end of the list. Any offset value less than '0' or greater than the length of the list needs no operation to be performed.
1 - Delete the element at the end of the list. No extra data is needed other than the operation code. No operation if the list is empty.
2 - Reverse the nodes of the list. No extra data is needed other than the operation code.
3 - Display elements of the list. No extra data is needed other than the operation code.
4 - Quit. No extra data is needed other than the operation code.

Output Format: Display is from the "Display" operation. A list is displayed with node values separated by spaces.

Sample Input:

```
0 10 0
0 20 2
0 20 0
0 30 0
3
0 40 2
0 50 1
3
2
3
1
1
3
4
```

Sample Output:

```
30 20 10
30 50 20 40 10
10 40 20 50 30
10 40 20
```

Lab Program 02: Implement a singly linked list with delete and other operations

The following operations are to be implemented in a Singly Linked List:
a) Delete the element at a specified position
b) Insert an element at the beginning
c) Reverse the nodes in the list
d) Display the elements of the list

Input Format: Every new line has one of the following operation code and any data needed for the operation with a ‘space’ in between. The program terminates on termination operation.

0 - Delete an element at the specified (0-based) position. In addition to the operation code, the 0-based position at which the element to be deleted is provided. Position '0' indicates the beginning of the list and the position value of one less than the length of the list indicates the last node of the list. Any offset value less than '0' or greater than or equal to the length of the list needs no operation to be performed.
1 - Insert the element at the beginning of the list. No extra data is needed other than the operation code.
2 - Reverse the nodes of the list. No extra data is needed other than the operation code.
3 - Display elements of the list. No extra data is needed other than the operation code.
4 - Quit. No extra data is needed other than the operation code.

Output Format: Display is from the "Display" operation. A list is displayed with node values separated by spaces.

Sample Input:

```
1 10
1 20
1 30
3
0 3
3
2
3
0 2
0 0
3
4
```

Sample Output:

```
30 20 10
30 20 10
10 20 30
20
```

Lab Program 03: Implement a doubly linked list with insert and other operations

The following operations are to be implemented in a Doubly Linked List:
a) Insert an element at a specified position
b) Delete the element at the end of the list
c) Display the elements in the list in reverse order
d) Display the elements of the list from the beginning

Input Format: Every new line has one of the following operation code and any data needed for the operation with a ‘space’ in between. The program terminates on termination operation.

0 - Insert an element at the specified position. In addition to the operation code, an element value and the offset after which the element to be inserted are provided. The element to be inserted after "offset" number of nodes. An offset value of '0' indicates the beginning of the list and an offset value of the length of the list indicates the end of the list. Any offset value less than '0' or greater than the length of the list needs no operation to be performed.
1 - Delete the element at the end of the list. No extra data is needed other than the operation code. No operation if the list is empty.
2 - Display the elements of the list in reversed order. No extra data is needed other than the operation code.
3 - Display elements of the list. No extra data is needed other than the operation code.
4 - Quit. No extra data is needed other than the operation code.

Output Format: Display is from the "Display" operations (operations codes 2 and 3). A list is displayed with node values separated by spaces.

Sample Input:
```
0 10 0
0 20 2
0 20 0
0 30 0
3
0 40 2
0 50 1
3
2
1
1
3
4
```

Sample Output:
```
30 20 10
30 50 20 40 10
10 40 20 50 30
30 50 20
```

Lab Program 04: Implement a doubly linked list with delete and other operations

The following operations are to be implemented in a Doubly Linked List:
a) Delete the element at a specified position
b) Insert an element at the beginning
c) Display the elements in the list in reverse order
d) Display the elements of the list from the beginning

Input Format: Every new line has one of the following operation code and any data needed for the operation with a ‘space’ in between. The program terminates on termination operation.

0 - Delete an element at the specified (0-based) position. In addition to the operation code, the 0-based position at which the element to be deleted is provided. Position '0' indicates the beginning of the list and the position of one less than the length of the list indicates the last node of the list. Any position value less than '0' or greater than or equal to the length of the list needs no operation to be performed.
1 - Insert the element at the beginning of the list. No extra data is needed other than the operation code.
2 - Display the elements of the list in reversed order. No extra data is needed other than the operation code.
3 - Display elements of the list. No extra data is needed other than the operation code.
4 - Quit. No extra data is needed other than the operation code.

Output Format: Display is from the "Display" operations (operations codes 2 and 3). A list is displayed with node values separated by spaces.

Sample Input:

```
1 10
1 20
1 30
3
0 3
3
2
3
1 40
0 2
0 0
3
4
```

Sample Output:

```
30 20 10
30 20 10
10 20 30
30 10
```

Lab Program 05: Implement a stack using a singly linked list

Implement a stack using a singly linked list to perform push, pop, peek and stack destroy operations.

Input Format: Every new line has one of the following operation code and any data needed for the operation with space in between.
0 - Insert an element at the top of the stack (Push operation). An element value is provided. There is no overflow case.
1 - Delete the element at the top of the stack (Pop operation). Print “Underflow” in case of an underflow case.
2 - Display the top element of the stack (Peek). Print “Empty Stack” in case of an empty stack.
3 - Destroy the stack and exit.

Output Format: Display is mostly from the Peek operation. Print the value in a new line. Print “Empty Stack” in case of an empty stack. In case of an underflow condition in Pop operation, “Underflow” to be printed.

Sample Input:
0 10
0 20
0 30
2
1
1
2
0 40
2
1
1
1
2
0 50
2
3

Sample Output:
30
10
40
Underflow
Empty Stack
50

Lab Program 06: Parentheses matching using stack data structure

Validate parentheses matching in an expression where everything except the three kinds of pairs of parentheses is stripped off. The three kinds of pairs of parentheses are “{“, ”}”, ”(“, ”)”, ”[“, and ”]”.

Input Format: A string from the alphabet of three pairs of parentheses.

Output Format: Print “Yes” if the parentheses match, “No” otherwise.

Sample Input 1:
`[()]{}{[()()]()}`

Sample Output 1:
Yes

Sample Input 2:
`((((()))){}[]`

Sample Output 2:
No

Lab Program 07: Infix to Postfix conversion

Convert an infix expression into the equivalent postfix expression.

Input Format: Infix expression is given in a single line. It is an ASCII string of an algebraic expression without spaces. Variables are of a single letter of the English alphabet and the allowed four operators are +, -, *, and /. Three kinds of pairs of parentheses are allowed; “(”, “)”, “{”, “}”, ”[”, and “]”.

Output Format: Print the equivalent postfix expression for the infix expression. Postfix expression must not have any parentheses.

Sample Input 1:
```
[{l+w*o}]+({z})/x
```

Sample Output 1:
```
lwo*+zx/+
```

Sample Input 2:
```
{l+w*o}*({{z}}/x)
```

Sample Output 2:
```
lwo*+zx/*
```

Sample Input 2:
```
({l+w*o})-[x+{z}]
```

Sample Output 2:
```
lwo*+xz+-
```

Lab Program 08: Implement a queue using a singly linked list

Implement a queue using a singly linked list to perform enqueue, dequeue, peek and queue destroy operations.

Input Format: Every new line has one of the following operation code and any data needed for the operation with space in between.
0 - Insert an element at the rear end of the queue (Enqueue operation). An element value is provided. There is no overflow case.
1 - Delete the element at the front end of the queue (Dequeue operation). Print “Underflow” in case of an underflow case.
2 - Display the front element of the queue (Peek). Print “Empty Queue” in case of an empty queue.
3 - Destroy the queue and exit.

Output Format: Display is mostly from the Peek operation. Print the value in a new line. Print “Empty Queue” in case of an empty queue. In case of an underflow condition in Dequeue operation, “Underflow” to be printed.

Sample Input:
0 10
0 20
0 30
2
1
1
2
0 40
2
1
1
1
2
0 50
2
3

Sample Output:
10
30
30
Underflow
Empty Queue
50

Lab Program 09: Implement a circular queue using an array

Implement a queue using an array to perform enqueue, dequeue, display the queue and queue destroy operations.

Input Format: The first line has n, the maximum number of entries the queue can have (1 ≤ n ≤ 100). Every new line has one of the following operation code and any data needed for the operation with space in between.
0 - Insert an element at the rear end of the queue (Enqueue operation). Print “Overflow” in case of an overflow case.
1 - Delete the element at the front end of the queue (Dequeue operation). Print “Underflow” in case of an underflow case.
2 - Display the queue in a single line from front to rear separated by spaces. Print “Empty Queue” in case of an empty queue.
3 - Destroy the queue and exit.

Output Format: Display is mostly from the “Display” operation. Print the queue from front to rear separated by spaces. Print “Empty Queue” in case of an empty queue. In case of an underflow condition in Dequeue operation, “Underflow” to be printed. In case of an overflow condition in Enqueue operation, “Overflow” to be printed.

Sample Input:

```
3
0 10
0 20
0 30
2
0 40
2
1
1
2
0 50
2
1
1
1
2
0 60
2
3
```

Sample Output:

```
10 20 30
Overflow
10 20 30
30
30 50
Underflow
Empty Queue
60
```

Lab Program 10: Circular deadly game

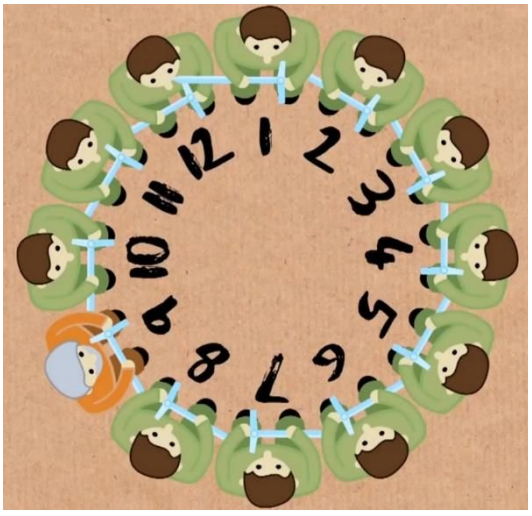
Let N people be numbered (ID) from **1** to **n** stand in a circle in order. Starting the count from **1**, we eliminate every second person until only one survivor is left. Find the people who were eliminated. Simulate the “deadly game” using a circular linked list.

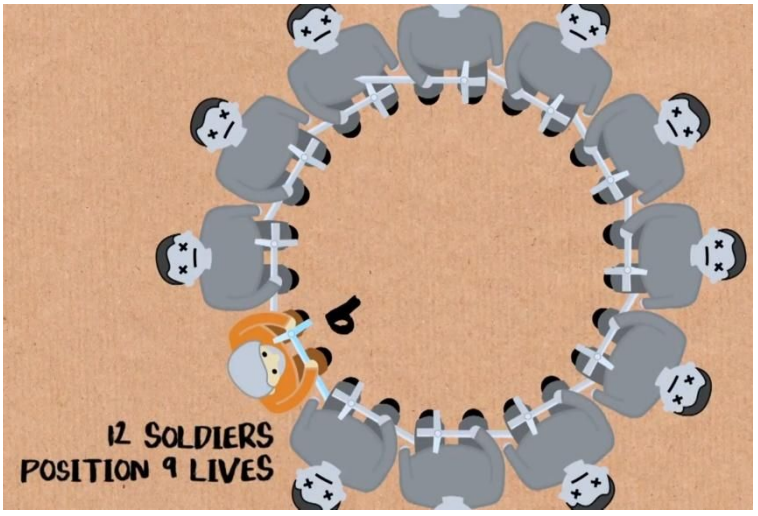
Input Format:
The first line contains the value of **n**.

Output Format:
Print the ID’s of people in the order they are executed in a single line with space separation.

Sample Input:
12

Sample Output:
2 4 6 8 10 12 3 7 11 5 1





Lab Program 11: Implement a queue using two stacks

Implement a queue using two stacks to perform enqueue, dequeue, peek and queue destroy operations.

Input Format: Every new line has one of the following operation code and any data needed for the operation with space in between.
0 - Insert an element at the rear end of the queue (Enqueue operation). An element value is provided. There is no overflow case.
1 - Delete the element at the front end of the queue (Dequeue operation). Print “Underflow” in case of an underflow case.
2 - Display the front element of the queue (Peek). Print “Empty Queue” in case of an empty queue.
3 - Destroy the queue and exit.

Output Format: Display is mostly from the Peek operation. Print the value in a new line. Print “Empty Queue” in case of an empty queue. In case of an underflow condition in Dequeue operation, “Underflow” to be printed.

Sample Input:
0 10
0 20
0 30
2
1
1
2
0 40
2
1
1
1
2
0 50
2
3

Sample Output:
10
30
30
Underflow
Empty Queue
50

Lab Program 12: Implement a BST and print the tree traversals

Implement a Binary Search Tree (BST) perform the inorder and postorder tree traversals.

Input Format: Every new line has one of the following operation code and any data needed for the operation with space in between.
0 - Insert an element into the BST.
1 - Print the values of the nodes of the inorder traversal separated by spaces.
2 - Print the values of the nodes of the postorder traversal separated by spaces.
3 - Destroy the BST and exit.

Output Format: Display is from the Print operations (operation codes 1 and 2).

Sample Input:
0 40
0 20
0 10
0 30
0 60
0 50
0 70
1
2
3

Sample Output:
10 20 30 40 50 60 70
10 30 20 50 70 60 40

Lab Program 13: Implement a BST and find the leaf and non-leaf count

Implement a Binary Search Tree (BST) and the find the number of leaf nodes and non-leaf nodes.

Input Format: Every new line has one of the following operation code and any data needed for the operation with space in between.
0 - Insert an element into the BST.
1 - Print the number of leaf nodes.
2 - Print the number of non-leaf nodes.
3 - Destroy the BST and exit.

Output Format: Display is from the Print operations (operation codes 1 and 2).

Sample Input:
0 40
0 20
0 10
0 30
0 60
1
0 50
0 70
2
3

Sample Output:
3
3

Lab Program 14: Construction of max-heap

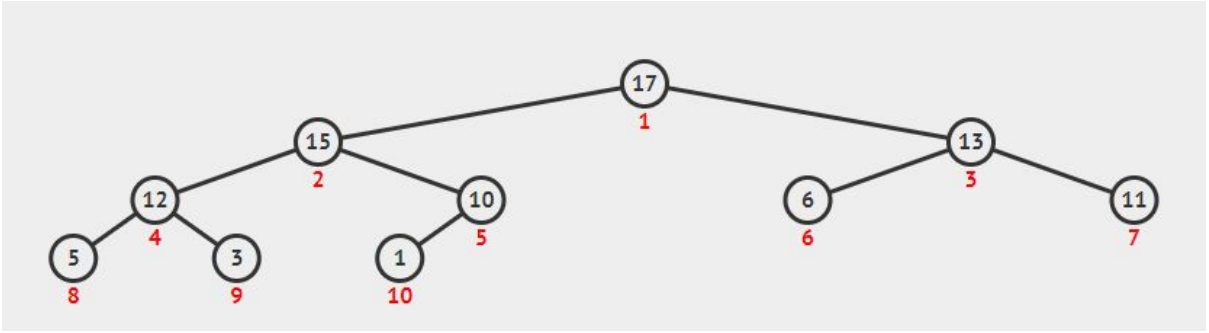
Construct a Max-Heap from the given sequence of key values and print the key values in the descending order.

Input Format:
The first line contains the value of **n**, the length of the input sequence of key values.
The next **n** lines contain key values to be inserted into the heap.

Output Format:
Print key values in descending order in a single line with space separation.

Sample Input:
10
5
6
10
12
13
17
11
15
3
1

Sample Output:
17 15 13 12 11 10 6 5 3 1



Lab Program 15: Implement Priority Queue using min-heap

Implement a priority queue where the minimum element is deleted using min-heap.

Input Format: The first line has n, the maximum number of entries the priority queue (min-heap) can have (1 ≤ n ≤ 100). Every new line has one of the following operation code and any data needed for the operation with space in between.
0 - Insert an element into the priority queue. An element value is provided. Print “Overflow” in case of overflow.
1 - Dequeue the minimum element the priority queue and print it. Print “Underflow” in case of an underflow case.
2 - Destroy the queue and exit.

Output Format: Display is mostly from the Dequeue operation. Print the value in a new line. Print “Underflow” in case of an empty queue. In case of an overflow condition in Insert operation, “Overflow” to be printed.

Sample Input:
3
1
0 50
0 15
0 25
0 100
1
1
0 55
0 30
1
2

Sample Output:
Underflow
Overflow
15
25
30

01	02	03
04	05	06
07	08	09
10	11	12
13	14	15