Data Structures Lab 7

Course: Data Structures (CL2001) **Semester:** Fall 2023
Instructor: Sameer Faisal **T.A:** N/A

Note:

- Lab manual cover following below Stack and Queue topics
  {Stack with Array and Linked list, Queue with Array and Linked, Priority Queue and its applications}
- Maintain discipline during the lab.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.
- Don't just blatantly copy the same code make changes to it accordingly

---

Stack with Array

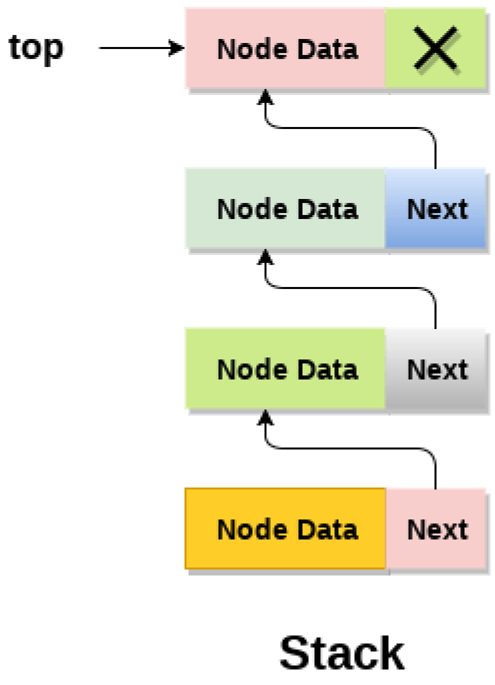
Sample Code of Stack in Array

```
class Stack {
    int top;
public:
    int a[MAX]; // Maximum size of Stack
    Stack() { top = -1; }
    bool push(int x);
    int pop();
    int peek();
    bool isEmpty();
};
bool Stack::push(int x)
{
    if (top >= (MAX - 1)) {
        cout << "Stack Overflow";
        return false;
    }
    else {
        a[++top] = x;
        cout << x << " pushed into stack\n";
        return true;
    }
}
int Stack::pop()
{
    if (top < 0) {
        cout << "Stack Underflow";
        return 0;
    }
    else {
        int x = a[top--];
        return x;
    }
}
int Stack::peek()
```

```
{
    if (top < 0) {
        cout << "Stack is Empty";
        return 0;
    }
    else {
        int x = a[top];
        return x;
    }
}

bool Stack::isEmpty()
{
    return (top < 0);
}
```

<u>Stack with Linked list</u>



Sample Code of Stack in Linked List

```
struct Node
{
   int data;
   struct Node* link;
};

struct Node* top;

// Utility function to add an element
// data in the stack insert at the beginning
void push(int data)
{

   // Create new node temp and allocate memory
   struct Node* temp;
   temp = new Node();

   // Check if stack (heap) is full.
   // Then inserting an element would
   // lead to stack overflow
   if (!temp)
   {
      cout << "\nHeap Overflow";
      exit(1);
   }

   // Initialize data into temp data field
   temp->data = data;

   // Put top pointer reference into temp link
   temp->link = top;

   // Make temp as top of Stack
   top = temp;
}
```

Example-1:

A. Design a Main class of upper code which perform the below task

1. Insert 10 Integers values in the stack
2. Write a utility function for upper code to display all the inserted integer values in the linked list in forward and reverse direction both
3. Write utility function to pop top element from the stack

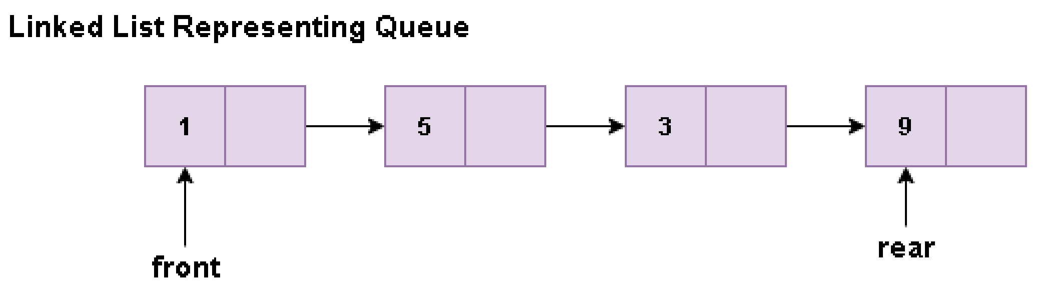
Queue with Array

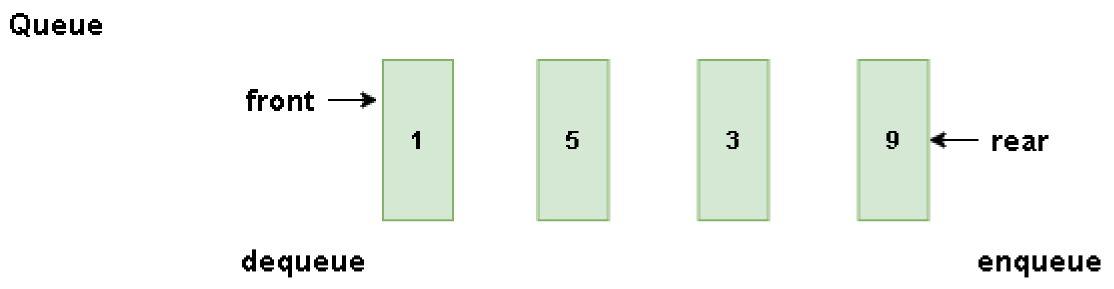
```
using namespace std;

// A structure to represent a queue
class Queue {
public:
    int front, rear, size;
    unsigned capacity;
    int* array;
};

// function to create a queue
// of given capacity.
// It initializes size of queue as 0
Queue* createQueue(unsigned capacity)
{
    Queue* queue = new Queue();
    queue->capacity = capacity;
    queue->front = queue->size = 0;
    // This is important, see the enqueue
    queue->rear = capacity - 1;
    queue->array = new int[queue->capacity];
    return queue;
}
```

Queue with Linked list





Sample Code

```
#include <iostream>
using namespace std;
struct node {
  int data;
  struct node *next;};
struct node* front = NULL;
struct node* rear = NULL;
struct node* temp;
void Insert() {
  int val;
  cout<<"Insert the element in queue : "<<endl;
  cin>>val;
  if (rear == NULL) {
    rear = (struct node *)malloc(sizeof(struct node));
    rear->next = NULL;
    rear->data = val;
    front = rear;
  } else {
    temp=(edlist struct node *)malloc(sizeof(struct node));
    rear->next = temp;
    temp->data = val;
    temp->next = NULL;
  rear = temp;
  }}
```

Priority Queue

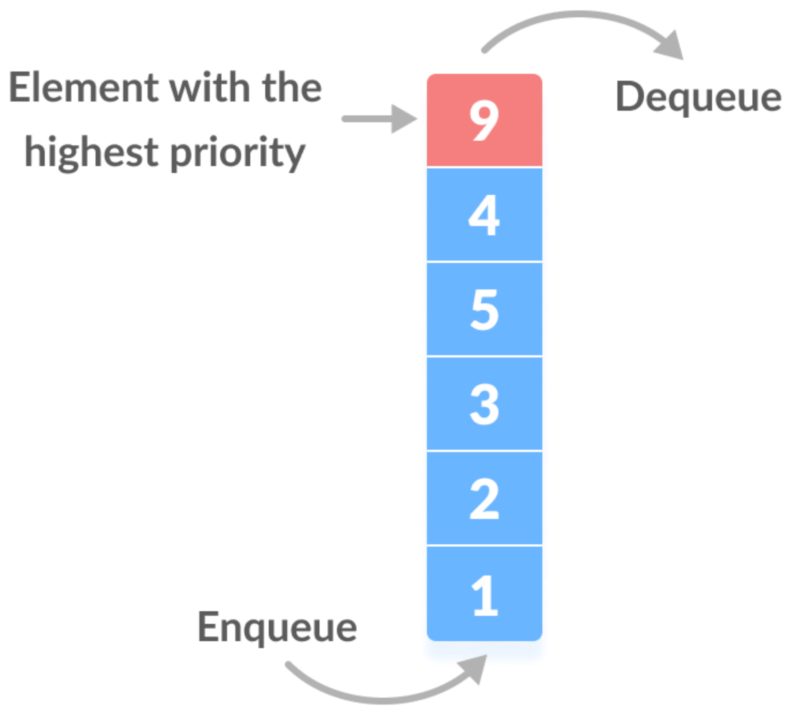
A priority queue is a **special type of queue** in which each element is associated with a **priority value**. And, elements are served on the basis of their priority. That is, higher priority elements are served first.

However, if elements with the same priority occur, they are served according to their order in the queue.

Assigning Priority Value

Generally, the value of the element itself is considered for assigning the priority. For example, The element with the highest value is considered the highest priority element. However, in other cases, we can assume the element with the lowest value as the highest priority element.

We can also set priorities according to our needs.



Removing Highest Priority Element

Difference between Priority Queue and Normal Queue
In a queue, the **first-in-first-out rule** is implemented whereas, in a priority queue, the values are removed **on the basis of priority**. The element with the highest priority is removed first.

Implementation of Priority Queue
Priority queue can be implemented using an array, a linked list, a heap data structure, or a binary search tree. Among these data structures, heap data structure provides an efficient implementation of priority queues.

Implementation Functions for priority Queue.

- **push():** This function is used to insert a new data into the queue.
- **pop():** This function removes the element with the highest priority from the queue.
- **peek() / top():** This function is used to get the highest priority element in the queue without removing it from the queue.

Algorithm :

PUSH(HEAD, DATA, PRIORITY):

```
• Step 1: Create new node with DATA and PRIORITY
• Step 2: Check if HEAD has lower priority. If true follow Steps 3-4 and end.
  Else goto Step 5.
• Step 3: NEW -> NEXT = HEAD
• Step 4: HEAD = NEW
• Step 5: Set TEMP to head of the list
• Step 6: While TEMP -> NEXT != NULL and TEMP -> NEXT -> PRIORITY
  > PRIORITY
• Step 7: TEMP = TEMP -> NEXT
  [END OF LOOP]
• Step 8: NEW -> NEXT = TEMP -> NEXT
• Step 9: TEMP -> NEXT = NEW
• Step 10: End
```

POP(HEAD):

```
• Step 1: Set the head of the list to the next node in the list. HEAD = HEAD ->
  NEXT.
• Step 2: Free the node at the head of the list
• Step 3: End
```

PEEK(HEAD):

```
• Step 1: Return HEAD -> DATA
• Step 2: End
```

Stack with Array and Linked list

- ➢ Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out) or FILO (First In Last Out).

  Mainly the following three basic operations are performed in the stack:

- **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- **Peek or Top:** Returns top element of stack.
- **isEmpty:** Returns true if stack is empty, else false

Application of Stack

- ➢ An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression. These notations are −

- Infix Notation
- Prefix (Polish) Notation
- Postfix (Reverse-Polish) Notation

Infix Notation

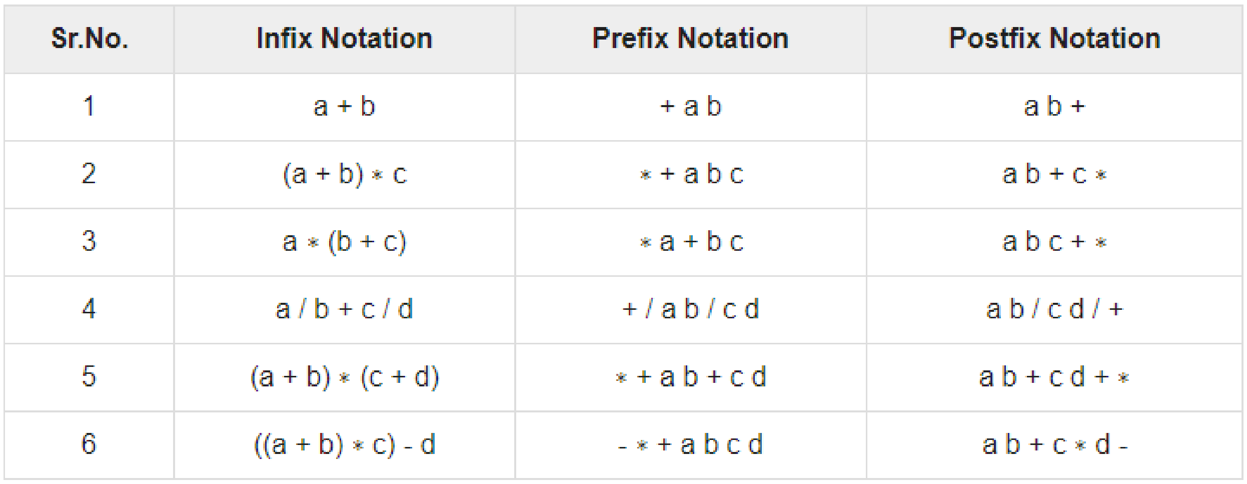
We write expression in **infix** notation, e.g. a - b + c, where operators are used **in**-between operands. It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices. An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.

Prefix Notation

In this notation, operator is **prefix**ed to operands, i.e. operator is written ahead of operands. For example, **+ab**. This is equivalent to its infix notation **a + b**. Prefix notation is also known as **Polish Notation**.

Postfix Notation

This notation style is known as **Reversed Polish Notation**. In this notation style, the operator is **postfix**ed to the operands i.e., the operator is written after the operands. For example, **ab+**. This is equivalent to its infix notation **a + b**.



Sr.No.	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) * c	* + a b c	a b + c *
3	a * (b + c)	* a + b c	a b c + *
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) * (c + d)	* + a b + c d	a b + c d + *
6	((a + b) * c) - d	- * + a b c d	a b + c * d -

Table-1

Example: In Table 1 Given above you are given an infix notation of ((a+b)*c)-d and you are required to convert it into postfix notation. Attempt to solve the question using queues and linked lists. (Assume a Tree when solving this approach)

Queue with Array and Linked list

- A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.
- **Following are the Operations on Queue**
  1.Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
  2.Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.
  3.Front: Get the front item from queue.
  4.Rear: Get the last item from queue.

Applications of Queue

- A priority queue in c++ is a type of container adapter, which processes only the highest priority element, i.e. the first element will be the maximum of all elements in the queue, and elements are in decreasing order.

Difference between a queue and priority queue :

- Priority Queue container processes the element with the highest priority, whereas no priority exists in a queue.
- Queue follows First-in-First-out (FIFO) rule, but in the priority queue highest priority element will be deleted first.
- If more than one element exists with the same priority, then, in this case, the order of queue will be taken.

1. **empty()** – This method checks whether the priority_queue container is empty or not. If it is empty, return true, else false. It does not take any parameter.

2. **size()** – This method gives the number of elements in the priority queue container. It returns the size in an integer. It does not take any parameter.

3. **push()** – This method inserts the element into the queue. Firstly, the element is added to the end of the queue, and simultaneously elements reorder themselves with priority. It takes value in the parameter.

4. **pop()** – This method delete the top element (highest priority) from the priority_queue. It does not take any parameter.

5. **top()** – This method gives the top element from the priority queue container. It does not take any parameter.

6. **swap()** – This method swaps the elements of a priority_queue with another priority_queue of the same size and type. It takes the priority queue in a parameter whose values need to be swapped.

7. **emplace()** – This method adds a new element in a container at the top of the priority queue. It takes value in a parameter.

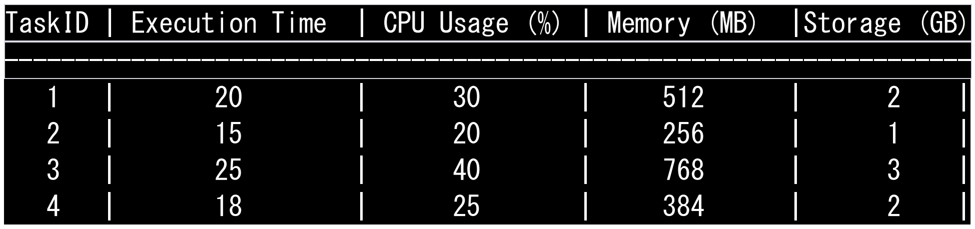
Lab Tasks:

Q1: Implement and insert the values “BORROWROB” in the stack and identify if it’s a palindrome or not. Use the push and pop functions to accomplish this (Note: Use Arrays to accomplish this)

Q2. Implement a Queue based approach where assume you are the cashier in a supermarket and you need to make checkouts. Customer ID’s Are 13,7,4,1,6,8,10. (Note: Use Arrays to accomplish this task with enqueue and dequeue)

Q3. Consider you have an expression x=12+13-5(0.5+0.5) +1 which results to 20. Implement a stack-based implementation to solve this question via linked lists (linked lists can be single or double) and the resulted output must be at the top of the stack. Note that the x and the equal sign must be present in the stack and when inserting the top value (20 result) all the values must be present in the stack (You can pop and push them accordingly)

Q5. In a cloud computing system, tasks are submitted by users for processing on a server. Each task is characterized by its estimated execution time (in minutes) and resource requirements, including CPU usage, memory, and storage. The server employs a queue-based job scheduling algorithm to manage the execution of these tasks efficiently.



TaskID	Execution Time	CPU Usage (%)	Memory (MB)	Storage (GB)
1	20	30	512	2
2	15	20	256	1
3	25	40	768	3
4	18	25	384	2

Assume the server has a single queue for processing tasks. Additionally, there are three virtual machines (VMs) available for task execution, each with the following resources:
CPU: 60%, Memory: 1024 MB, Storage: 5 GB. Build a priority queue based approach using linked-lists and dequeue the elements in such a way that the resources are utilized optimally and all the tasks are executed.