Data Structures and Algorithms Lab

Application of Stacks and Queues

Subject Code: 17ECSP201 **Lab No:** 05 **Semester:** III

Date: Oct 2017 **Batch:** MSM

Question: Application of Stacks and Queues

Objective: Understanding the usage of stacks and queues in real time scenarios

---

Problem 01:

Hubli city is definitely on a right path while talking towards the city development. To name a few are road construction, city maintenance or building a drainage system etc. Works are being carried out everywhere. Apart from these, good or bad, the city is also slowly adapting to the technology and rural culture. Examples are U-Mall, Urban OASIS Mall, Laxmi Pride Cinemas, KFC, Dominos, etc, etc.

Urban Oasis Mall, is preparing a master plan on how to attract more customers and kind of shops that needs to be started in the remaining spaces of the mall. Also it wants the process to get automated.

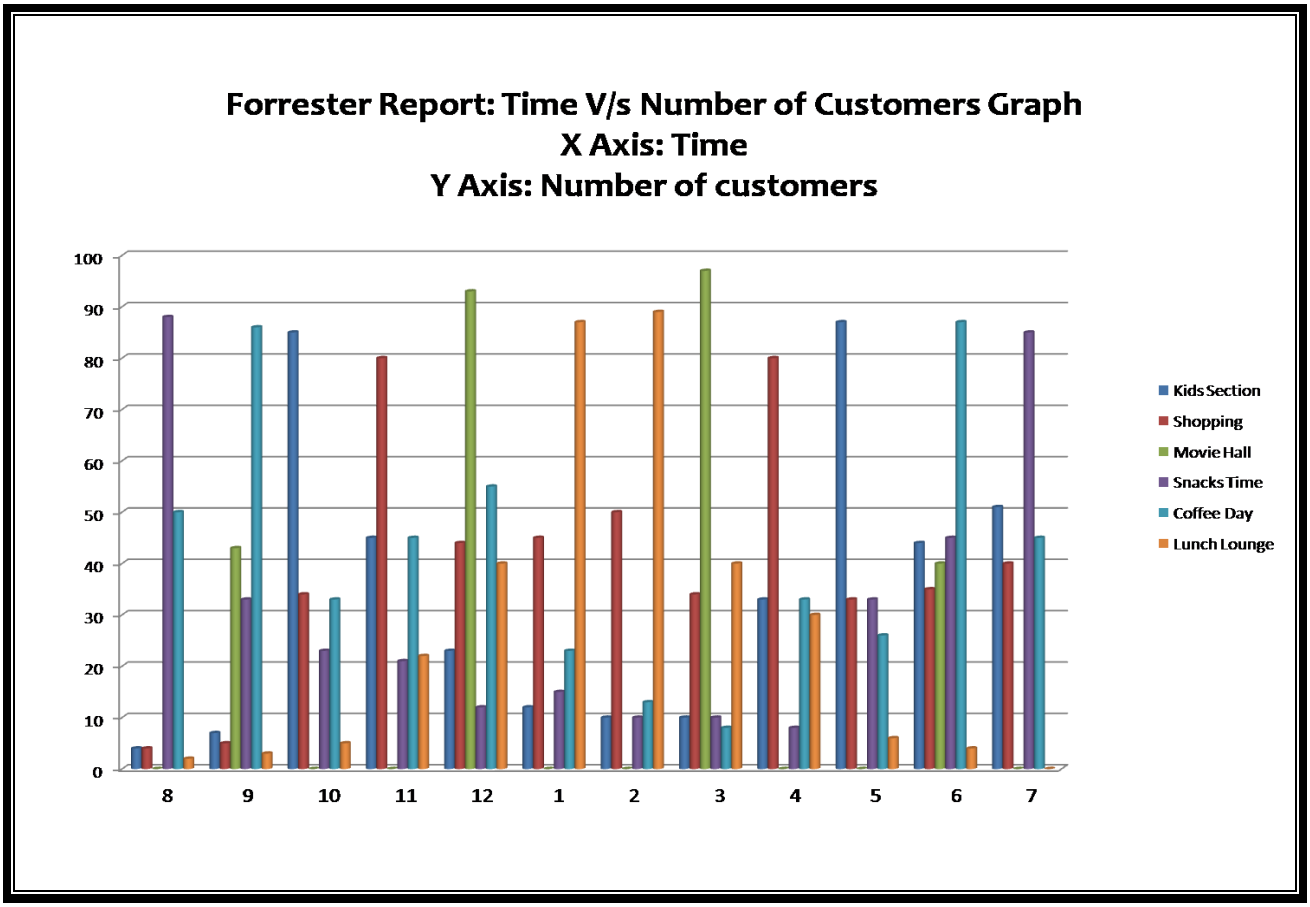
Fig: Urban OASIS Mall, Hubli City

[Image Reference: ebharat.com]

As the first initiative, the mall wants to attract more customers. And the technique adapted is to send the messages every hour to all the mobiles detected in the nearby locality. (Spanning 1km radius)

It was necessary to know the user habits at other malls in order to decide what kind of message would be appropriate to be sent at every hour. The authorities had approached the **Forrester Report** to get the survey done!

Forrester Reports collected the stats for number of customers present at each hour in 20 different malls of India. And after a study for 1 year, here is the **Forrester Report** summary:



According to the survey analysis, at each shop there are more customers at the specified timings:

Kids Section: 10.00 and 5.00
Shopping: 11.00 and 4.00
Movie Hall: 12.00 and 3.00
Snacks Time: 08.00 and 7.00
Coffee Day: 9.00 and 6.00
Lunch Lounge: 1.00 and 2.00

Here is the peak flow of customers at every hour:

8.00 - Snacks Time

9.00 – Coffee Day

10.00 – Kids Section

11.00 – Shopping

12.00 – Movie Hall

1.00 – Lunch Lounge

2.00 – Lunch Lounge

3.00 – Movie Hall

4. 00 – Shopping

5.00 – Kids Section

6.00 – Coffee Day

7.00 – Snacks Time

Do you observe any pattern above??

At Professor's Desk

After looking at the message pattern, **STACK** looks like the best suitable data structure for the application. But we will have to maintain 2 stacks. At every hour a pop from stack will be push to another stack.

Can you justify why stack is the best one?

A little more help to you.

Use the below structures:

```
struct advertisement
{
  char message_sent[30];      // Mobile advertisement to be made
  int time[2];                // Time of broadcast
};

struct stack
{
  int top;                                  // The stack  top
  int active;                               // Status to indicate if the stack is active
  struct advertisement items[STACKSIZE];    //  Items tracking advertisements
```

};

typedef struct stack STACK;

Provide the functionalities in main for:

1. Display the message being broadcasted
2. Update Time
3. Print messages left over
4. Print All Messages
5. Exit

Task Description:

1. **Display the message being broadcasted –** Peek into the active stack
2. **Update Time -** Pop the value from one stack and push it to another stack. The advertisement message is being changed
3. **Print messages left over -** Find the active stack and display all the contents
4. **Print All Messages** – Print both the stack contents clearly stating the type of stack
5. **Exit –** Exit from the menu

Initialize the data in stack statically before providing the above options to the user. Before the above menu is being printed to the user, the initialization function needs to be called which will initialize one stack with respected content and keep the other empty.

Split the code into 3 different files. Test for all pre and post conditions. Some part of the code is already done for you. You need to do the remaining.

Wait!
That is not all of it.
There is another problem that needs to be solved.

Problem 02:

Gartner Report for this year has made an interesting prediction about producer-consumer problem. Looking at the different advancements and researches and mainly motivated from the nature, the report says:

"Producer is no more only a producer and so about the consumer"

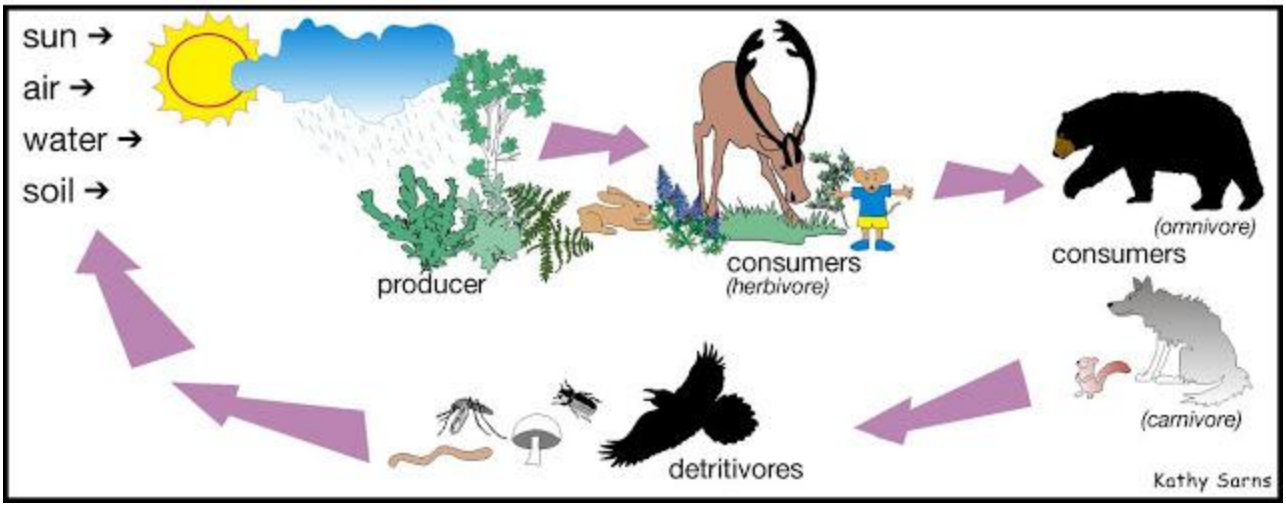


Fig: The Producer – Consumer Food chain

[Image Reference: chapeijin.blogspot.com]

Well,
For an engineer that is too much of information. You can start coding!

At Professor's Desk

The problem has left behind many open things. Let us make all valid assumptions and move ahead with coding.

Our task is to simulate a producer which will produce as well as consume and so about consumer.

Let us visualize and implement the new prediction.

Any doubts in why we are selecting a queue to implement it?

Use the below structures:

```
struct indices
{
  int front1;
  int rear1;
  int front2;
  int rear2;
  int produce;
  int consume;
};

struct prod_cons
{
  int data[MAXQUEUE];
  struct indices index;
};
```

typedef struct prod_cons PC;

Provide the functionalities in main for:

1. Produce in Producer
2. Consume from Consumer
3. Produce in Consumer
4. Consume from Producer
5. Print Producer-Consumer
6. Get the Active Producer and Consumer
7. Exit

Task Description:

1. **Produce In Producer –** Enqueue a random data into the producer queue
2. **Consume from Consumer –** Dequeue the data from consumer queue
3. **Produce in Consumer –** Enqueue a random data into a consumer queue
4. **Consume from Producer –** Dequeue the data from producer queue
5. **Print Producer Consumer –** Print the details present in both the queue with appropriate messages
6. **Get the Active Producer and Consumer –** Print in which queue the last enqueue and dequeue has happened
7. **Exit –** Exit from Menu

Points of Discussion:

- How are you going to visualize producer and consumer in integer array 'data'?
- What will be the empty condition?
- What will be the full condition?
- How will you generate a random integer data? Of what range?

Split the code into 3 different files. Test for all pre and post conditions.

**** Happy Coding ****