Indian Statistical Institute
Semester-I 2023–2024
M.Tech.(CS) - First Year
Lab Test 1 (8 November, 2023)
Subject: Computing Laboratory

Total: 65 marks **Maximum marks: 60** **Duration: 3 hrs.**

INSTRUCTIONS

1. You may consult or use slides / programs provided to you as course material, or programs that you have written yourself as part of classwork / homework for this course, but please **do not** consult or use material from other Internet sources, your classmates, or anyone else.
2. Please make sure that your programs adhere strictly to the specified input and output format. **Your program may not pass the test cases provided, if your program violates the input and output requirements.**
3. Submissions from different students having significant match will be **debarred from evaluation**.
4. You may use C / Python for this test.

NOTE: Unless otherwise specified, all programs should take the required inputs from stdin, and print the desired outputs to stdout.

Q1. Write a program to create a data structure named StackCouple that represents the implementation of two stacks. Implementation of StackCouple should use ONLY a single array, i.e., both stacks should use the same array for storing elements. Two suggested possibilities are as follows:

- Odd-even strategy: In this case, Stack 1 uses locations 0,2,4,... of the array, whereas Stack 2 uses the array locations 1,3,5,....
- Colliding strategy: In this case, the two stacks start from the two ends of the array and grow in opposite directions (towards one another).

Implement both the strategies. Refer to the sample program structure provided.

Complete the main function to read in a sequence of operations from stdin adhering to the following convention. The first line of the input is either 0 / 1 to specify the strategy that the program will work with. If option is 0, you should use the odd-even strategy. Otherwise, use the colliding strategy. Following this, read in a sequence of operations, wherein each operation is as below: 0/1 1/2 character (optional). The first entry specifies a push / pop (0 for push and 1 for pop). The second entry specifies the stack identifier (1 for Stack 1, 2 for Stack 2). If the first entry is a push, you have a character as the third entry, and nothing otherwise. The input is terminated with -1. [10 + 10]

```
Sample Input-1
1
0 1 p
0 2 Q
0 1 h
0 1 e
0 1 v
0 1 a
0 2 M
0 1 p
0 2 B
0 1 n
0 1 k
0 1 a
0 1 g
0 1 w
0 1 f
1 2
0 1 g
0 1 v
1 1
0 1 j
1 2
0 1 t
0 1 r
1 1
0 1 d
0 1 n
0 2 G
1 2
1 1
1 2
0 2 U
1 2
0 1 n
0 1 j
0 1 a
0 1 e
0 2 L
0 1 n
0 2 R
0 1 b
```

```
 0 1 h
 0 1 k
-1
```

Sample Output-1:

```
p________________________
p_______________________Q
ph______________________Q
phe_____________________Q
phev____________________Q
pheva___________________Q
pheva__________________MQ
phevap_________________MQ
phevap________________BMQ
phevapn_______________BMQ
phevapnk______________BMQ
phevapnka_____________BMQ
phevapnkag____________BMQ
phevapnkagw___________BMQ
phevapnkagwf__________BMQ
phevapnkagwf___________MQ
phevapnkagwfg__________MQ
phevapnkagwfgv_________MQ
phevapnkagwfg__________MQ
phevapnkagwfgj_________MQ
phevapnkagwfgj__________Q
phevapnkagwfgjt_________Q
phevapnkagwfgjtr________Q
phevapnkagwfgjt_________Q
phevapnkagwfgjtd________Q
phevapnkagwfgjtdn_______Q
phevapnkagwfgjtdn______GQ
phevapnkagwfgjtdn_______Q
phevapnkagwfgjtd________Q
phevapnkagwfgjtd_________
phevapnkagwfgjtd________U
phevapnkagwfgjtd_________
phevapnkagwfgjtdn________
phevapnkagwfgjtdnj_______
phevapnkagwfgjtdnja______
phevapnkagwfgjtdnjae_____
phevapnkagwfgjtdnjae____L
```

```
phevapnkagwfgjtdnjaen___L
phevapnkagwfgjtdnjaen__RL
phevapnkagwfgjtdnjaenb_RL
phevapnkagwfgjtdnjaenbhRL
Error: Overflow in stack.
```

```
Sample Input-2:
0
0 1 e
0 2 N
0 1 q
0 1 p
0 2 E
0 1 p
0 1 j
0 1 u
0 2 S
0 2 I
0 2 I
0 1 m
1 1
0 2 O
1 1
0 1 q
1 1
0 2 Z
0 1 v
0 1 f
0 1 n
1 1
1 1
0 1 k
0 1 n
1 2
0 1 u
0 2 U
0 1 y
0 1 l
0 1 f
0 1 o
0 1 n
-1
```

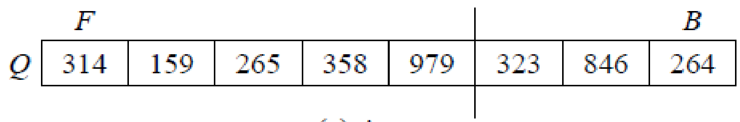
```
Sample Output-2
e________________________
eN_______________________
eNq______________________
eNq_p____________________
eNqEp____________________
eNqEp_p__________________
eNqEp_p_j________________
eNqEp_p_j_u______________
eNqEpSp_j_u______________
eNqEpSpIj_u______________
eNqEpSpIjIu______________
eNqEpSpIjIu_m____________
eNqEpSpIjIu______________
eNqEpSpIjIuO_____________
eNqEpSpIjI_O_____________
eNqEpSpIjIqO_____________
eNqEpSpIjI_O_____________
eNqEpSpIjI_O_Z___________
eNqEpSpIjIvO_Z___________
eNqEpSpIjIvOfZ___________
eNqEpSpIjIvOfZn__________
eNqEpSpIjIvOfZ___________
eNqEpSpIjIvO_Z___________
eNqEpSpIjIvOkZ___________
eNqEpSpIjIvOkZn__________
eNqEpSpIjIvOk_n__________
eNqEpSpIjIvOk_n_u________
eNqEpSpIjIvOkUn_u________
eNqEpSpIjIvOkUn_u_y______
eNqEpSpIjIvOkUn_u_y_l____
eNqEpSpIjIvOkUn_u_y_l_f__
eNqEpSpIjIvOkUn_u_y_l_f_o
Error: Overflow in stack.

Sample Input-3
1
0 2 Q
0 1 x
0 1 b
0 1 e
```

```
0 1 v
0 1 e
0 1 v
0 1 s
0 1 i
1 2
0 1 b
0 1 t
1 2
-1
```

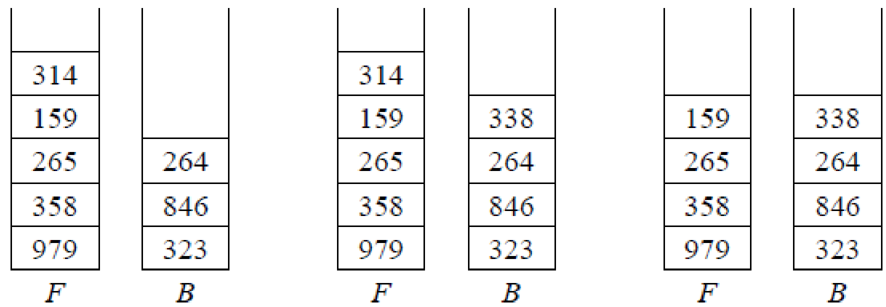
```
Sample Output-3
________________________Q
x_______________________Q
xb______________________Q
xbe_____________________Q
xbev____________________Q
xbeve___________________Q
xbevev__________________Q
xbevevs_________________Q
xbevevsi________________Q
xbevevsi_________________
xbevevsib________________
xbevevsibt_______________
Error: Underflow in stack.
```

Q2. Consider the implementation of a queue using 2 stacks. As shown in the following figure, a queue Q (see Part (a) of the figure) can be realized with two stacks F and B (see Part (b)). An arbitrary break-point is chosen (between 979 and 323 in the figure). The part of Q before this break-point resides in the front stack F, and the part of Q after the break-point resides in the back stack B. Notice the order in which the elements of Q appear in F and B.

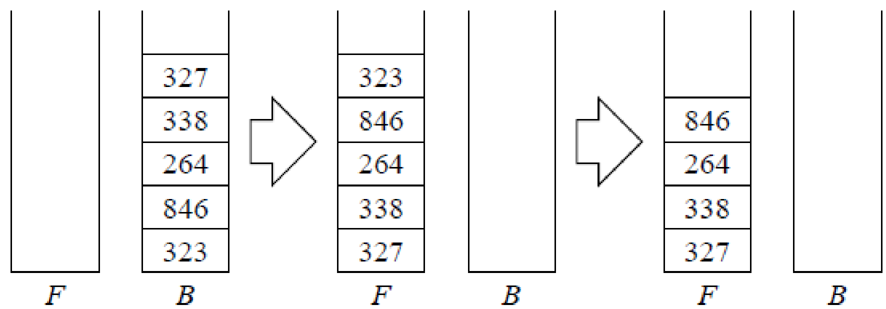
An enqueue operation involves pushing the new item to the back stack B (see Part (c)). A dequeue operation is the same as pop from the front stack F. If F was not empty before the pop, this is straightforward (see Part (d)). If both F and B are empty, then Q is empty too, and a dequeue from Q is not permitted. If F is empty but B contains one or more elements (see Part (e)), the element to dequeue lies at the bottom of B, and cannot be directly accessed. Make a sequence of pop operations from B and push operations of those elements to F, until B becomes empty. Now, a normal dequeue (pop from F) can be performed.



(a) A queue



(b) Queue in two stacks (c) Enqueue (d) Dequeue



(e) Dequeue from an empty front stack

Write a program to implement a queue this way. You are allowed to use the implementation of a stack data structure that you have. You are NOT ALLOWED TO USE ANY ADDITIONAL ARRAYS OR LISTS, only the 2 stacks should suffice. You may define your queue as:

```
typedef struct {
      STACK F, B;
} QUEUE;
```

For a stack S, the following functions are needed. Only stacks of integers are supported.

- S = SINIT(): Create an empty stack.
- ISEMPTY(S): Returns 1 or 0 depending on whether S is empty or not.
- TOP(S): Returns the element (an integer) at the top of S.
- S = PUSH(S, x): Push an integer x to the stack S.
- S = POP(S): Perform a pop operation from S.
- SPRNT2B(S): Print the elements of S from top to bottom.
- SPRNB2T(S): Print the elements of S from bottom to top.

You do not need to reimplement the Stack operations if they are already available with you, implement only the ones that you need. Use the above calls to implement the queue data structure as follows.

- Q = QINIT(): Create an empty queue.
- Q = ENQUEUE(Q, x): Enqueue an integer x to Q.
- Q = DEQUEUE(Q): Perform a dequeue operation on Q.
- QPRN(Q) Print the elements of Q from front to back.

Write a main routine to read in a sequence of operations from stdin, where each line is specified as: 0/1 (0 for enqueue and 1 for dequeue). If it is an enqueue operation, you have an additional integer to enqueue, whereas nothing is specified in case of a dequeue operation. The input sequence is terminated with -1.

[10]

```
Sample Input-1
0 250
1
-1
Sample Output-1
Q = [ 250 ]
Q = [ ]

Sample Input-2:
0 505
1
0 149
0 736
0 939
0 373
0 172
0 317
0 649
0 166
1
1
0 143
1
1
1
1
1
```

```
1
1
-1

Sample Output-2:
Q = [ 505 ]
Q = [ ]
Q = [ 149 ]
Q = [ 149 736 ]
Q = [ 149 736 939 ]
Q = [ 149 736 939 373 ]
Q = [ 149 736 939 373 172 ]
Q = [ 149 736 939 373 172 317 ]
Q = [ 149 736 939 373 172 317 649 ]
Q = [ 149 736 939 373 172 317 649 166 ]
Q = [ 736 939 373 172 317 649 166 ]
Q = [ 939 373 172 317 649 166 ]
Q = [ 939 373 172 317 649 166 143 ]
Q = [ 373 172 317 649 166 143 ]
Q = [ 172 317 649 166 143 ]
Q = [ 317 649 166 143 ]
Q = [ 649 166 143 ]
Q = [ 166 143 ]
Q = [ 143 ]
Q = [ ]
```

Q3. Let M be a positive integer and A $= \{0, 1, 2, 3, ..., M-1\}$ the set of all remainders modulo M. We start with a random element x_0 of A. Subsequently, for i = 1,2,3,..., we generate $x_i = (x_{i-1}^2 + 1)\%M$. Since the elements of the sequence x_0, x_1, x_2, $\ldots$ are from the finite set A, there must be a match x_i $= x_j$ after finitely many iterations. After that, the sequence is periodic, since every element of the sequence is uniquely determined only by the previous element. Your task is to use a singly linked list to represent this. Start with the following standard type definition.

```
typedef struct _node {
      int data;            /* value stored in a node */
      struct _node *next;  /* pointer to the next node */
} node;
```

Write a program that has the following functions:

(a) A function `genList()` with the prototype `node *genrho ( int M , int x )` that accepts the modulus M and the initial value x_0. It creates a list using the scheme described above, and returns a pointer to the header node of the list. Do not use a dummy node at the beginning of the list.

(b) A function `cyclelen()` with the prototype `int cyclelen ( node *head )` that accepts, as its only parameter, a pointer to the header of such a list and returns the length of the cycle of the list. It is important to mention that the number of nodes or any such auxiliary information must not be generated during the creation of the list. The function cyclelen only assumes that the header pointer points to a valid list.

(c) A main function to report the output of your program on the following parameters.

- M = 100, x = 5
- M = 6543, x = 3456
- M = 35791, x = 13579

[4+8+3]

```
Sample Input-Output:

M = 50 and x = 11.
11: Inserted... continuing...
22: Inserted... continuing...
35: Inserted... continuing...
26: Inserted... continuing...
27: Inserted... continuing...
30: Inserted... continuing...
1: Inserted... continuing...
2: Inserted... continuing...
5: Inserted... continuing...
26: Cycle detected... breaking...
M = 50, x = 11, cycle length = 6
```

Q4. Consider a list of N customers who visit a photocopy shop to get various documents copied. The list consists of $N+1$ lines. The first line contains the positive integer N. Each of the remaining N lines contains 2 fields: the arrival time of a customer (in HH:MM format, where $00 \leq \text{HH} \leq 23$, $00 \leq \text{MM} \leq 59$) and the time required (in minutes) to complete her copy job. Assume that the i-th line corresponds to customer C_i ($1 \leq i \leq N$). Note that the list will, in general, **not** be sorted by arrival times. Assume that the shop has a single photocopy machine, operated by a single person, who uses a first-come-first-served (FCFS) scheduling algorithm. In other words, customers' jobs are taken up in the order in which they arrive.

(a) NOTE: On the submission portal, Q4(a) corresponds to **question number 4**.

Write a function that takes a time string of the form `HH:MM` as input, and converts it into the number of minutes since the beginning of the day (`00:00`). For example, given the string `00:23`, your function should return the integer 23; similarly, the return value for `10:15` should be 615; the return value for `14:32` should be 872. Your function should have the following prototype:

```
int hh_mm_to_minutes(char *);
```

If the input string is not of the form `HH:MM`, your function should return -1.

Test your function by writing a program that reads from stdin a list of customers (in the format specified above), and for each customer, prints the arrival time in minutes since the beginning of the day on a separate line. [5]

(b) NOTE: On the submission portal, Q4(b) corresponds to **question number 5**.

Write a program that takes a list of customers in the format specified above via stdin, and prints the **order** in which the customers are served. If the arrival times of two customers are the same, then the customers are served in the order in which they are listed in the input file (i.e., C_i is served before C_j if and only if $i < j$). [5]

(c) NOTE: On the submission portal, Q4(c) corresponds to **question number 6**.

Define the *waiting time* of a customer as the difference (in minutes) between the time when she arrives at the shop, and the time when the operator **STARTS** working on her job.

Write a program to print the customers along with their waiting times, in descending order of waiting time. [10]