Informatics II
Exercise 7 / **Solution**

April 06, 2020

Abstract Data Types (ADT)

Task 1. Create a new datatype `twoStack` of positive integers that represents two stacks. This datatype should use only one array i.e., both stacks should use the same array for storing elements. A stack is of the following type:

```
1 typedef struct stackADT {
2     int *arr;
3     int size;
4     int top1, top2;
5 } twoStack;
```

Your implementation should start two stacks from two extreme corners of arr[]. The `stack1` starts from the leftmost element, the first element in `stack1` is pushed at index 0. The `stack2` starts from the rightmost corner, the first element in `stack2` is pushed at index (size-1). Both stacks grow and shrink in opposite direction.

Along with the above datatype create the following functions:
There are two ways to implement `top` of the stack. The `top` can either point to

1. The first free element of the stack.
2. or the top element of the stack.

Both approaches are equally good and valid as well. In our solution, the `top` points to the top element of the stack.

- `void initialize(twoStack *s, int n)` which initializes stack `s`.

```
1 void initialize(twoStack *s, int n) {
2     s->size = n;
3     s->arr = malloc(n*sizeof(int));
4     s->top1 = -1;
5     s->top2 = n;
6 }
```

- `void push1(twoStack *s, int value)` which inserts element `value` into first stack.

```
1 void push1(twoStack *s, int value){
2     if (s->top1 < s->top2-1){
3         s->top1++;
4         s->arr[s->top1] = value;
5     }
6     else{
7         printf("%s\n", "Stack Overflow");
8     }
9 }
```

- `void push2(twoStack *s, int value)` which inserts element `value` into second stack.

```
1 void push2(twoStack *s, int value){
2     if (s->top1 < s->top2-1){
3         s->top2--;
4         s->arr[s->top2] = value;
5     }
6     else{
7         printf("%s\n", "Stack Overflow");
8     }
9 }
```

- `int pop1(twoStack *s)` which removes an element from first stack and returns its value. In case of an error, -1 should be returned.

```
1 int pop1(twoStack *s){
2     if (s->top1 >=0 ){
3           int x = s->arr[s->top1];
4           s->top1--;
5           return x;
6       }
7       else{
8           printf("%s\n", "Stack Underflow");
9           return -1;
10      }
11 }
```

- `int pop2(twoStack *s)` which removes an element from second stack and returns its value. In case of an error, -1 should be returned.

```
1 int pop2(twoStack *s){
2     if (s->top2 < s->size){
3           int x = s->arr[s->top2];
4           s->top2++;
5           return x;
6       }
7       else{
8           printf("%s\n", "Stack Underflow");
9           return -1;
10      }
11 }
```

- `void printStack(twoStack *s)` that prints the values of the elements of the twoStack to the console enclosed in brackets, e.g. `[ 3 5 7 2 ]`.

```
1  void printStack(twoStack *s) {
2      if (s->size > 0) {
3          int i;
4          printf("[ ");
5          for (i = 0; i <=s->size - 1; i++) {
6              printf("%d ", s->arr[i]);
7          }
8          printf("]\n");
9      }
10 }
```

Test your implementation by performing the following operations:

1. Create and initialize stack s with array size of 5.
2. Insert 5 into first stack
3. Insert 10, 15 into second stack
4. Insert 11 into first stack
5. Insert 7 into second stack
6. Print the elements on the console
7. Remove one element of first stack and print the value of the removed element.
8. Remove one element of second stack and print the value of the removed element.
9. Insert 27 into first stack
10. Insert 92 into the second stack
11. Print the elements on the console

```
1   twoStack *s = malloc(sizeof(twoStack));
2   initialize(s, 5);
3   push1(s, 5);
4   push2(s, 10);
5   push2(s, 15);
6   push1(s, 11);
7   push2(s, 7);
8   push2(s, 8);
9   printStack(s);
10  printf("Pop one from first stack: %d\n", pop1(s));
11  printf("Pop one from second stack: %d\n", pop2(s));
12  push1(s, 27);
13  push2(s, 92);
14  printStack(s);
```

Task 2. Consider a datatype `queue` corresponding to a circular queue of positive integers which be implemented using Circular linked list. A queue is of the following structure:

```
1 struct queue {
2   struct node* head;
3   struct node* tail;
4 };
```

```
1 struct node {
2   int data;
3   struct node* next;
4 };
```

Considering the `queue` datatype write the C code for the following functions:

- `struct queue* initialize()` which initializes the `head` and the `tail` nodes of `q`.

```
1 struct queue* initialize() {
2     struct queue* q = malloc(sizeof(struct queue));
3
4     q->head = NULL;
5     q->tail = NULL;
6     return q;
7 }
```

- `void enQueue(queue *q, int value)` which inserts the element `value` in `q`.

```
1 void enQueue(struct queue *q, int value) {
2     struct node *temp = malloc(sizeof(struct node));;
3     temp->data = value;
4     if (q->head == NULL)
5         q->head = temp;
6     else
7         q->tail->next = temp;
8
9     q->tail = temp;
10    q->tail->next = q->head;
11 }
```

- `int deQueue(queue *q)` which removes an element from `q` and returns its value. In case of an error, -1 should be returned.

```
1 int deQueue(struct queue *q) {
2     if (q->head == NULL) {
3         printf ("Queue is empty\n");
4         return -1;
5     }
6
7     // If this is the last node to be deleted
8     int value; // Value to be dequeued
9     if (q->head == q->tail) {
10        value = q->head->data;
11        free(q->head);
12        q->head = NULL;
```

```
13          q->tail = NULL;
14      }
15      // There are more than one nodes
16      else {
17          struct node *temp = q->head;
18          value = temp->data;
19          q->head = q->head->next;
20          q->tail->next= q->head;
21          free(temp);
22      }
23
24      return value ;
25  }
```

- `void displayQueue(struct queue *q)` that prints the values of the elements of the cicular queue to the console enclosed in brackets, e.g. `[ 3 5 7 2 ]`.

```
1  void displayQueue(struct queue *q) {
2      struct node *temp = q->head;
3      printf("[ ");
4      while (temp->next != q->head) {
5          printf("%d ", temp->data);
6          temp = temp->next;
7      }
8      printf("%d ]\n", temp->data);
9  }
```

After you have implemented and tested these functions with lists of your choice, include the following sequence in your *main()* method:

1. Insert 14, 22, 6 in the queue `q`.
2. Print all the elemets of queue `q`.
3. Dequeue two elements and print their values.
4. Print `q`.
5. Enqueue 9, 20 in the queue `q`
6. Print `q`

```
1      // Create a queue and initialize front and rear
2      struct queue* q = initialize();
3
4      // Inserting elements in Circular Queue
5      enQueue(q, 14);
6      enQueue(q, 22);
7      enQueue(q, 6);
8
9      // Display elements present in Circular Queue
10     displayQueue(q);
11
```

```
12      // Deleting elements from Circular Queue
13      printf("Deleted value = %d\n", deQueue(q));
14      printf("Deleted value = %d\n", deQueue(q));
15
16      // Remaining elements in Circular Queue
17      displayQueue(q);
18
19      enQueue(q, 9);
20      enQueue(q, 20);
21      displayQueue(q);
```