Informatics II, Spring 2023, Solution 7

Publication of exercise: April 17, 2023
Publication of solution: April 23, 2023
Exercise classes: April 24 - 28, 2023

Task 1.

1. PUSH(4) — 4
   PUSH(1) — 4 1
   PUSH(3) — 4 1 3
   POP() — 4 1
   PUSH(8) — 4 1 8
   POP() — 4 1

2. ENQUEUE(4) — 4
   ENQUEUE(1) — 4 1
   ENQUEUE(3) — 4 1 3
   DEQUEUE() — 1 3
   ENQUEUE(8) — 1 3 8
   DEQUEUE() — 3 8

3. The first stack starts at 1 and grows up towards n, while the second starts from n and decreases to 1. Stack overflow happens when an element is pushed when the two stack pointers are adjacent.

4. ENQUEUE: $\Theta(1)$.
   DEQUEUE: worst $O(n)$, amortized $\Theta(1)$(on average).
   Let the two stacks be A and B.
   ENQUEUE pushes elements on B. ENQUEUE is always $\Theta(1)$. DEQUEUE pops elements from A. If A is empty, the contents of B are transferred to A by popping them out of B and pushing them to A. That way, they appear in reverse order and are popped in the original order.
   DEQUEUE operation can perform in $\Theta(n)$ time, but that will happen only when A is empty. If many ENQUEUEs and DEQUEUEs are performed, the total time will be linear to the number of elements. For example, we ENQUEUE n elements and DEQUEUE n elements. All n elements are transferred from B to A only once, so in total n times. The amortized complexity of DEQUEUE = n / n = 1, which is $\Theta(1)$(on average).

5. PUSH: $\Theta(1)$.
   POP: $\Theta(n)$.
   We have two queues – q_1 and q_2. PUSH operation always enqueues elements in q_1. Assume that q_1 contains i elements: e_1,...,e_i. POP operation: (1) dequeue e_1,...,e_{i-1} elements and remain element e in q_1 (2) enqueue e_1,...,e_{i-1} in order to q_2. (3) dequeue e from q_1 and return e.
   The PUSH operation is $\Theta(1)$. The POP operation is $\Theta(n)$ where n is the number of elements in the stack. In other words, there are n elements in q_1.

Task 2.

1.

```c
#include <stdio.h>
#define SIZE 10

int stack[SIZE];
int top = -1;

void push(int value)
{
    if(top<SIZE-1)
    {
        if (top < 0)
        {
            stack[0] = value;
            top = 0;
        }
        else
        {
            stack[top+1] = value;
            top++;
        }
    }
    else
    {
        printf("Stackoverflow!!!!\n");
    }
}

int isempty()
{
    return top<0;
}

int pop()
{
    if(!isempty())
    {
        int n = stack[top];
        top--;
        return n;
    }
    else
    {
        printf("Error: the stack is empty!\n");
        return -99999;
    }
}

int Top()
{
    if (!isempty())
    {
        return stack[top];
    }
    else
    {
        printf("Error: the stack is empty!\n");
        return -99999;
    }
}

void display()
{
    int i;
    for(i=0;i<=top;i++)
    {
```

```
66          printf("%d,",stack[i]);
67      }
68      printf("\n");
69  }
70
71  int main()
72  {
73      push(4);
74      push(8);
75      printf("isempty:␣%d\n", isempty());
76      printf("Top:␣%d\n", Top());
77      display();
78
79      pop();
80      printf("\nisempty:␣%d\n", isempty());
81      printf("Top:␣%d\n", Top());
82      display();
83
84      pop();
85      printf("\nisempty:␣%d\n", isempty());
86      printf("Top:␣%d\n", Top());
87      display();
88
89      pop();
90
91      return 0;
92  }
```

2.

```c
#include <stdio.h>
#define MAXSIZE 10

int queue[MAXSIZE];

int front = -1;
int rear = -1;
int size = -1;

int isempty()
{
    return size <= 0;
}

int isfull()
{
    return size == MAXSIZE;
}

void enqueue(int value)
{
    if(size<MAXSIZE)
    {
        if(isempty())
        {
            queue[0] = value;
            front = rear = 0;
            size = 1;
        }
        else if(rear == MAXSIZE-1)
        {
            queue[0] = value;
            rear = 0;
            size++;
        }
        else
        {
            queue[rear+1] = value;
            rear++;
            size++;
        }
    }
    else
    {
        printf("Queue is full\n");
    }
}

int Front()
{
    if(isempty())
    {
        printf("Queue is empty\n");
        return -1;
    }
    else
    {
        return queue[front];
    }
}

int dequeue()
{
    int ret = Front();
    size--;
    front++;
    if (front == MAXSIZE) {
        front = 0;
```

```
69      }
70      return ret;
71  }
72
73  void display()
74  {
75      if(isempty())
76      {
77          printf("Queue is empty\n");
78          return;
79      }
80
81      int i;
82      if(rear>=front)
83      {
84          for(i=front;i<=rear;i++)
85          {
86              printf("%d,",queue[i]);
87          }
88      }
89      else
90      {
91          for(i=front;i<MAXSIZE;i++)
92          {
93              printf("%d,",queue[i]);
94          }
95          for(i=0;i<=rear;i++)
96          {
97              printf("%d,",queue[i]);
98          }
99      }
100         printf("\n");
101 }
102
103 int main()
104 {
105     display();
106     enqueue(4);
107     enqueue(8);
108     enqueue(10);
109     enqueue(20);
110     display();
111     dequeue();
112     printf("After dequeue\n");
113     display();
114     enqueue(50);
115     enqueue(60);
116     enqueue(70);
117     enqueue(80);
118     dequeue();
119     enqueue(90);
120     enqueue(100);
121     enqueue(110);
122     enqueue(120);
123     enqueue(130);
124     enqueue(140);
125     enqueue(150);
126     printf("After enqueue\n");
127     display();
128     dequeue();
129     printf("After dequeue\n");
130     display();
131     enqueue(160);
132     printf("After enqueue\n");
133     display();
134     return 0;
135 }
```

3.

```c
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0

struct node
{
    int data;
    struct node *next;
};
typedef struct node node;

node *top;

void initialize()
{
    top = NULL;
}

void push(int value)
{
    node *tmp;
    tmp = malloc(sizeof(node));
    tmp -> data = value;
    tmp -> next = top;
    top = tmp;
}

int pop()
{
    node *tmp;
    int n;
    tmp = top;
    n = tmp->data;
    top = top->next;
    free(tmp);
    return n;
}

int Top()
{
    return top->data;
}

int isempty()
{
    return top==NULL;
}

void display(node *head)
{
    if(head == NULL)
    {
        printf("NULL\n");
    }
    else
    {
        printf("%d,", head -> data);
        display(head->next);
    }
}

int main()
{
    initialize();
    push(10);
    push(20);
    push(30);
```

```
69      printf("The␣top␣is␣%d\n",Top());
70      pop();
71      printf("The␣top␣after␣pop␣is␣%d\n",Top());
72      display(top);
73      return 0;
74  }
```

4.

```c
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
#define FULL 10

struct node
{
    int data;
    struct node *next;
};
typedef struct node node;

struct queue
{
    int count;
    node *front;
    node *rear;
};
typedef struct queue queue;

void initialize(queue *q)
{
    q->count = 0;
    q->front = NULL;
    q->rear = NULL;
}

int isempty(queue *q)
{
    return (q->rear == NULL);
}

void enqueue(queue *q, int value)
{
    if (q->count < FULL)
    {
        node *tmp;
        tmp = malloc(sizeof(node));
        tmp->data = value;
        tmp->next = NULL;
        if(!isempty(q))
        {
            q->rear->next = tmp;
            q->rear = tmp;
        }
        else
        {
            q->front = q->rear = tmp;
        }
        q->count++;
    }
    else
    {
        printf("List␣is␣full\n");
    }
}

int dequeue(queue *q)
{
    node *tmp;
    int n = q->front->data;
    tmp = q->front;
    q->front = q->front->next;
    q->count--;
    free(tmp);
    return(n);
}
```

```
69
70 void display(node *head)
71 {
72     if(head == NULL)
73     {
74         printf("NULL\n");
75     }
76     else
77     {
78         printf("%d,", head -> data);
79         display(head->next);
80     }
81 }
82
83 int main()
84 {
85     queue *q;
86     q = malloc(sizeof(queue));
87     initialize(q);
88     enqueue(q,10);
89     enqueue(q,20);
90     enqueue(q,30);
91     printf("Queue␣before␣dequeue\n");
92     display(q->front);
93     dequeue(q);
94     printf("Queue␣after␣dequeue\n");
95     display(q->front);
96     return 0;
97 }
```

Task 3.

Algorithm: REVERSEEVEN(Q)

```
S = initStack()
qSize = queueSize(Q)
for i = 1 to qSize do
    val = deQueue(Q)
    if val %2 ==0 then
        push(S,val)
    enQueue(val)
for i = 1 to qSize do
    val = deQueue(Q)
    if val %2 ==0 then
        enQueue(Q,pop(S))
    else
        enQueue(val)
```