

Informatics II

Exercise 7

April 12, 2021

Goals:

- Practise operations of stacks and queues
- Implement stacks and queues using arrays
- Implement stacks and queues using linked lists
- Study and practise Deque

Abstract Data Types: Stacks, Queues

Task 1. Abstract structures of stacks and queues

- a) Illustrate the result of each operation in the sequence PUSH(4), PUSH(1), PUSH(3), POP(), PUSH(8), and POP(S) on an initially empty stack S.

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

- b) Illustrate the result of each operation in the sequence ENQUEUE(4), ENQUEUE(1), ENQUEUE(3), DEQUEUE(), ENQUEUE(8), and DEQUEUE() on an initially empty queue Q.

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

- c) Explain how to implement two stacks in one array $A[]$ in such a way that neither stack overflows unless the total number of elements in both stacks together is n . The PUSH and POP operations should run in $O(1)$ time.

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.

- d) Explain how to implement a queue using two stacks. Analyze the running time of the queue operations.

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).

- e) Explain how to implement a stack using two queues. Analyze the running time of the stack operations.

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, \dots, e_i . POP operation: (1) dequeue e_1, \dots, e_{i-1} elements and remain element e in q_1 (2) enqueue e_1, \dots, 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. Implementation of stacks and queues in C

- a) Write a C program that implements a stack using an array. Your C program should contain push and pop functions, and examples to call implemented functions.

```

1 #include <stdio.h>
2 #define SIZE 10
3
4 int stack[SIZE];
5 int top = -1;
6
7 void push(int value)
8 {
9     if(top < SIZE-1)
10     {
11         if (top < 0)
12         {
13             stack[0] = value;
14             top = 0;
15         }
16         else
17         {
18             stack[top+1] = value;
19             top++;
20         }
21     }
22     else
23     {
24         printf("Stackoverflow!!!!\n");
25     }

```

```
26 }
27
28 int isempty()
29 {
30     return top<0;
31 }
32
33 int pop()
34 {
35     if(!isempty())
36     {
37         int n = stack[top];
38         top--;
39         return n;
40     }
41     else
42     {
43         printf("Error:_the_stack_is_empty!\n");
44         return -99999;
45     }
46 }
47
48 int Top()
49 {
50     if (isempty())
51     {
52         return stack[top];
53     }
54     else
55     {
56         printf("Error:_the_stack_is_empty!\n");
57         return -99999;
58     }
59 }
60
61 void display()
62 {
63     int i;
64     for(i=0;i≤top;i++)
65     {
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 }
```

- b) Write a C program that implements a queue using an array. Your C program should contain enqueue and dequeue functions, and examples to call implemented functions.

```
1  #include <stdio.h>
2  #define MAXSIZE 10
3
4  int queue[MAXSIZE];
5
6  int front = -1;
7  int rear = -1;
8  int size = -1;
9
10 int isempty()
11 {
12     return size ≤ 0;
13 }
14
15 int isfull()
16 {
17     return size == MAXSIZE;
18 }
19
20 void enqueue(int value)
21 {
22     if(size < MAXSIZE)
23     {
24         if(isempty())
25         {
26             queue[0] = value;
27             front = rear = 0;
28             size = 1;
29         }
30         else if(rear == MAXSIZE-1)
31         {
32             queue[0] = value;
33             rear = 0;
34             size++;
35         }
36         else
37         {
38             queue[rear+1] = value;
39             rear++;
40             size++;
41         }
42     }
```

```
42     }
43     else
44     {
45         printf("Queue_is_full\n");
46     }
47 }
48
49 int Front()
50 {
51     if(isempty())
52     {
53         printf("Queue_is_empty\n");
54         return -1;
55     }
56     else
57     {
58         return queue[front];
59     }
60 }
61
62 int dequeue()
63 {
64     int ret = Front();
65     size--;
66     front++;
67     if (front == MAXSIZE) {
68         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 }
```

- c) Write a C program that implements a stack using a singly linked list. The operations PUSH and POP should still take $O(1)$ time.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #define TRUE 1
4  #define FALSE 0
5
6  struct node
7  {
8      int data;
9      struct node *next;
10 };
11 typedef struct node node;
12
13 node *top;
14
```

```
15 void initialize()
16 {
17     top = NULL;
18 }
19
20 void push(int value)
21 {
22     node *tmp;
23     tmp = malloc(sizeof(node));
24     tmp->data = value;
25     tmp->next = top;
26     top = tmp;
27 }
28
29 int pop()
30 {
31     node *tmp;
32     int n;
33     tmp = top;
34     n = tmp->data;
35     top = tmp->next;
36     free(tmp);
37     return n;
38 }
39
40 int Top()
41 {
42     return top->data;
43 }
44
45 int isempty()
46 {
47     return top==NULL;
48 }
49
50 void display(node *head)
51 {
52     if(head == NULL)
53     {
54         printf("NULL\n");
55     }
56     else
57     {
58         printf("%d,", head->data);
59         display(head->next);
60     }
61 }
62
63 int main()
64 {
65     initialize();
66     push(10);
67     push(20);
68     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 }
```

- d) Write a C program that implements a queue using a singly linked list. The operations ENQUEUE and DEQUEUE should still take $O(1)$ time.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define TRUE 1
4  #define FALSE 0
5  #define FULL 10
6
7  struct node
8  {
9      int data;
10     struct node *next;
11 };
12 typedef struct node node;
13
14 struct queue
15 {
16     int count;
17     node *front;
18     node *rear;
19 };
20 typedef struct queue queue;
21
22 void initialize(queue *q)
23 {
24     q->count = 0;
25     q->front = NULL;
26     q->rear = NULL;
27 }
28
29 int isempty(queue *q)
30 {
31     return (q->rear == NULL);
32 }
33
34 void enqueue(queue *q, int value)
35 {
36     if (q->count < FULL)
37     {
38         node *tmp;
39         tmp = malloc(sizeof(node));
40         tmp->data = value;
41         tmp->next = NULL;
42         if (!isempty(q))
43         {
44             q->rear->next = tmp;
45             q->rear = tmp;
46         }
47         else
48         {
```



```
49         q->front = q->rear = tmp;
50     }
51     q->count++;
52 }
53 else
54 {
55     printf("List is full\n");
56 }
57 }
58
59 int dequeue(queue *q)
60 {
61     node *tmp;
62     int n = q->front->data;
63     tmp = q->front;
64     q->front = q->front->next;
65     q->count--;
66     free(tmp);
67     return(n);
68 }
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 }
```

e) Comparing stacks and queues using linked lists and stacks and queues using arrays.

- Implementations using arrays has the limitation of size. If we fixed the array, the stack and queue have a limited capacity. If we resize the array, we need to create a new array.
- Implementations using linked lists don't have the limitation of fixed size, because elements are added and removed through pointers.

- Implementations using arrays require less space compared the implementation using linked lists, because linked lists require additional pointers.

Task 3. A double-ended queue, abbreviated to deque, allows elements added to the front and removed from the rear. We use an array of integers as the data structure for a deque of integers. Write a C program that contains the following functions:

1. addFront(), add an integer to the front
2. addRear(), add an integer to the rear
3. delFront(), remove an integer from the front
4. delRear(), remove an integer from the rear.

All four functions should have time complexity of $O(1)$. Consider how to implement these four functions. Your C program should contain examples to call these four implemented functions.

```
1  /* C Program to implement Deque using circular array */
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #define MAX 7
6
7  int deque_arr[MAX];
8  int front = -1;
9  int rear = -1;
10
11 void insert_frontEnd(int item);
12 void insert_rearEnd(int item);
13 int delete_frontEnd();
14 int delete_rearEnd();
15 void display();
16 int isEmpty();
17 int isFull();
18
19 int main()
20 {
21     insert_rearEnd(5);
22     insert_frontEnd(12);
23     insert_rearEnd(11);
24     insert_frontEnd(5);
25     insert_rearEnd(6);
26     insert_frontEnd(8);
27
28     printf("\nElements in a deque:");
29     display();
30
31     int i = delete_frontEnd();
32     printf("\nremoved item: %d", i);
33
34     printf("\nElements in a deque after deletion:");
35     display();
36
37     insert_rearEnd(16);
38     insert_rearEnd(7);
```

```
39
40 printf("\nElements in a deque after addition:");
41 display();
42
43 printf("\nInsert when full:");
44 insert_rearEnd(7);
45
46 i = delete_rearEnd();
47 printf("\nremoved item: %d", i);
48
49 printf("\nElements in a deque after deletion:");
50 display();
51 } /*End of main()*/
52
53 void insert_frontEnd(int item)
54 {
55     if (isFull())
56     {
57         printf("\nQueue Overflow\n");
58         return;
59     }
60     if (front == -1) /*If queue is initially empty*/
61     {
62         front = 0;
63         rear = 0;
64     }
65     else if (front == 0)
66         front = MAX - 1;
67     else
68         front = front - 1;
69     deque_arr[front] = item;
70 } /*End of insert_frontEnd()*/
71
72 void insert_rearEnd(int item)
73 {
74     if (isFull())
75     {
76         printf("\nQueue Overflow\n");
77         return;
78     }
79     if (front == -1) /*if queue is initially empty*/
80     {
81         front = 0;
82         rear = 0;
83     }
84     else if (rear == MAX - 1) /*rear is at last position of queue */
85         rear = 0;
86     else
87         rear = rear + 1;
88     deque_arr[rear] = item;
89 } /*End of insert_rearEnd()*/
90
91 int delete_frontEnd()
92 {
93     int item;
94     if (isEmpty())
```

```
95  {
96    printf("\nQueue_Underflow\n");
97    exit(1);
98  }
99  item = deque_arr[front];
100  if (front == rear) /*Queue has only one element */
101  {
102    front = -1;
103    rear = -1;
104  }
105  else if (front == MAX - 1)
106    front = 0;
107  else
108    front = front + 1;
109  return item;
110 } /*End of delete_frontEnd()*/
111
112 int delete_rearEnd()
113 {
114   int item;
115   if (isEmpty())
116   {
117     printf("\nQueue_Underflow\n");
118     exit(1);
119   }
120   item = deque_arr[rear];
121
122   if (front == rear) /*queue has only one element*/
123   {
124     front = -1;
125     rear = -1;
126   }
127   else if (rear == 0)
128     rear = MAX - 1;
129   else
130     rear = rear - 1;
131   return item;
132 } /*End of delete_rearEnd() */
133
134 int isFull()
135 {
136   if ((front == 0 && rear == MAX - 1) || (front == rear + 1))
137     return 1;
138   else
139     return 0;
140 } /*End of isFull()*/
141
142 int isEmpty()
143 {
144   if (front == -1)
145     return 1;
146   else
147     return 0;
148 } /*End of isEmpty()*/
149
150 void display()
```

```
151 {  
152     int i;  
153     if (isEmpty())  
154     {  
155         printf("\nQueue_is_empty\n");  
156         return;  
157     }  
158     printf("\nQueue_elements:\n");  
159     i = front;  
160     if (front ≤ rear)  
161     {  
162         while (i ≤ rear)  
163             printf("%d_", deque_arr[i++]);  
164     }  
165     else  
166     {  
167         while (i ≤ MAX - 1)  
168             printf("%d_", deque_arr[i++]);  
169         i = 0;  
170         while (i ≤ rear)  
171             printf("%d_", deque_arr[i++]);  
172     }  
173     printf("\n");  
174 } /*End of display() */
```