University of Zurich

Department of Informatics
Database Technology Group
Prof. Dr. Anton Dignös

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 decreses 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 transfered 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,...,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. 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.

```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++)
    {
        printf("%d,",stack[i]);
    }
    printf("\n");
}

int main()
{
    push(4);
    push(8);
    printf("isempty: %d\n", isempty());
    printf("Top: %d\n", Top());
    display();

    pop();
    printf("\nisempty: %d\n", isempty());
    printf("Top: %d\n", Top());
```

```c
    display();

    pop();
    printf("\nisempty: %d\n", isempty());
    printf("Top: %d\n", Top());
    display();

    pop();

    return 0;
}
```

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.

```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;
    }
    return ret;
}

void display()
{
    if(isempty())
    {
        printf("Queue␣is␣empty\n");
        return;
    }

    int i;
    if(rear≥front)
    {
        for(i=front;i≤rear;i++)
        {
            printf("%d,",queue[i]);
        }
    }
    else
    {
        for(i=front;i<MAXSIZE;i++)
        {
            printf("%d,",queue[i]);
        }
        for(i=0;i≤rear;i++)
        {
            printf("%d,",queue[i]);
```

```
        }
    }
        printf("\n");
}

int main()
{
    display();
    enqueue(4);
    enqueue(8);
    enqueue(10);
    enqueue(20);
    display();
    dequeue();
    printf("After dequeue\n");
    display();
    enqueue(50);
    enqueue(60);
    enqueue(70);
    enqueue(80);
    dequeue();
    enqueue(90);
    enqueue(100);
    enqueue(110);
    enqueue(120);
    enqueue(130);
    enqueue(140);
    enqueue(150);
    printf("After enqueue\n");
    display();
    dequeue();
    printf("After dequeue\n");
    display();
    enqueue(160);
    printf("After enqueue\n");
    display();
    return 0;
}
```

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.

```
#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;

```

```c
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);
    printf("The top is %d\n",Top());
    pop();
```

```c
    printf("The top after pop is %d\n",Top());
    display(top);
    return 0;
}
```

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.

```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
        {
```

```c
            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);
}

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

int main()
{
    queue *q;
    q = malloc(sizeof(queue));
    initialize(q);
    enqueue(q,10);
    enqueue(q,20);
    enqueue(q,30);
    printf("Queue before dequeue\n");
    display(q->front);
    dequeue(q);
    printf("Queue after dequeue\n");
    display(q->front);
    return 0;
}
```

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.

```c
/* C Program to implement Deque using circular array */

#include <stdio.h>
#include <stdlib.h>
#define MAX 7

int deque_arr[MAX];
int front = -1;
int rear = -1;

void insert_frontEnd(int item);
void insert_rearEnd(int item);
int delete_frontEnd();
int delete_rearEnd();
void display();
int isEmpty();
int isFull();

int main()
{
  insert_rearEnd(5);
  insert_frontEnd(12);
  insert_rearEnd(11);
  insert_frontEnd(5);
  insert_rearEnd(6);
  insert_frontEnd(8);

  printf("\nElements in a deque: ");
  display();

  int i = delete_frontEnd();
  printf("\nremoved item: %d", i);

  printf("\nElements in a deque after deletion: ");
  display();

  insert_rearEnd(16);
  insert_rearEnd(7);
```

```

    printf("\nElements in a deque after addition: ");
    display();

    printf("\nInsert when full: ");
    insert_rearEnd(7);

    i = delete_rearEnd();
    printf("\nremoved item: %d", i);

    printf("\nElements in a deque after deletion: ");
    display();
} /*End of main()*/

void insert_frontEnd(int item)
{
    if (isFull())
    {
        printf("\nQueue Overflow\n");
        return;
    }
    if (front == -1) /*If queue is initially empty*/
    {
        front = 0;
        rear = 0;
    }
    else if (front == 0)
        front = MAX - 1;
    else
        front = front - 1;
    deque_arr[front] = item;
} /*End of insert_frontEnd()*/

void insert_rearEnd(int item)
{
    if (isFull())
    {
        printf("\nQueue Overflow\n");
        return;
    }
    if (front == -1) /*if queue is initially empty*/
    {
        front = 0;
        rear = 0;
    }
    else if (rear == MAX - 1) /*rear is at last position of queue */
        rear = 0;
    else
        rear = rear + 1;
    deque_arr[rear] = item;
} /*End of insert_rearEnd()*/

int delete_frontEnd()
{
    int item;
    if (isEmpty())
```

```
  {
    printf("\nQueue_Underflow\n");
    exit(1);
  }
  item = deque_arr[front];
  if (front == rear) /*Queue has only one element */
  {
    front = −1;
    rear = −1;
  }
  else if (front == MAX − 1)
    front = 0;
  else
    front = front + 1;
  return item;
} /*End of delete_frontEnd()*/

int delete_rearEnd()
{
  int item;
  if (isEmpty())
  {
    printf("\nQueue_Underflow\n");
    exit(1);
  }
  item = deque_arr[rear];

  if (front == rear) /*queue has only one element*/
  {
    front = −1;
    rear = −1;
  }
  else if (rear == 0)
    rear = MAX − 1;
  else
    rear = rear − 1;
  return item;
} /*End of delete_rearEnd() */

int isFull()
{
  if ((front == 0 && rear == MAX − 1) || (front == rear + 1))
    return 1;
  else
    return 0;
} /*End of isFull()*/

int isEmpty()
{
  if (front == −1)
    return 1;
  else
    return 0;
} /*End of isEmpty()*/

void display()
```

```
{
  int i;
  if (isEmpty())
  {
    printf("\nQueue is empty\n");
    return;
  }
  printf("\nQueue elements :\n");
  i = front;
  if (front <= rear)
  {
    while (i <= rear)
      printf("%d ", deque_arr[i++]);
  }
  else
  {
    while (i <= MAX - 1)
      printf("%d ", deque_arr[i++]);
    i = 0;
    while (i <= rear)
      printf("%d ", deque_arr[i++]);
  }
  printf("\n");
} /*End of display() */
```