SDF-2 Tut-5

Q1)1. The five items: A, B, C, D, and E are pushed in a stack, one after other starting from

A. The stack is popped four items and each element is inserted in a queue. The two

elements are deleted from the queue and pushed back on the stack. Now one item is

popped from the stack. The popped item is?

D

Q2)

How many stacks are needed to implement a queue? Consider the situation where no

other data structure like arrays, linked list is available to you.

2 stacks

Q3)

How many queues are needed to implement a stack? Consider the situation where no

other data structure like arrays, linked list is available to you.

2 Queues

Q4)4. WAP to reverse the first K elements of a Queue

Example: Input: Q = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100] and k = 5

Output: Q = [50, 40, 30, 20, 10, 60, 70, 80, 90, 100]

Program to reverse First K elements of queue:

```
#include<stdio.h>

#include<stdlib.h>

struct stackrecord

{

int *array;

int capacity;

int tos;

};

typedef struct stackrecord *stack;

stack createstack(int max)

{

stack s;

s=malloc(sizeof(struct stackrecord));

if(s==NULL)

{

printf("out of space");

}

s->array=malloc((sizeof(int))*max);

if(s->array==NULL)

{

printf("out of space");

}

s->capacity=max-1;
```

```
s->tos=-1;
return(s);
}
int isemptys(stack s)
{
return s->tos==-1;
}
int isfulls(stack s)
{
return s->tos==s->capacity;
}
void push(int x,stack s)
{
if(isfulls(s))
printf("Overflow");
else
{
printf("\n %d is pushed",x);
s->tos++;
s->array[s->tos]=x;
}
}
int topandpop(stack s)
{
if(isemptys(s))
```

```
{
printf("\n empty stack");
return;
}
else
{
printf("\n %d is popped",s->array[s->tos]);
return s->array[s->tos--];
}
}
struct queuerecord
{
int *array;
int front;
int rear;
int capacity;
};
typedef struct queuerecord *queue;
queue createqueue(int max)
{
queue q;
q=malloc(sizeof(struct queuerecord));
if(q==NULL)
printf("Error");
q->array=malloc(sizeof(int)*max);
```

```
if(q->array==NULL)
printf("Error");
q->capacity=max-1;
q->front=-1;
q->rear=-1;
return q;
}
int isfullq(queue q)
{
return (q->rear==q->capacity);
}
int isemptyq(queue q)
{
return (q->front==-1);
}
void enqueue(queue q,int x)
{
if(isfullq(q))
printf("overflows");
else
{
printf("\n %d is enqueued",x);
q->rear++;
q->array[q->rear]=x;
if(q->front==-1)
```

```
Q->FRONT++;
}
}
INT FRONTANDDELETE(QUEUE Q)
{
INT P;
IF(ISEMPTYQ(Q))
{
PRINTF("UNDERFLOW");
RETURN;
}
ELSE
{
P=Q->ARRAY[Q->FRONT];
PRINTF("\N %D IS FRONT AND DELETED",P);
Q->FRONT++;
RETURN P;
}
}
VOID DISPLAY(QUEUE Q)
{
INT I;
IF(ISEMPTYQ(Q))
{
PRINTF("UNDERFLOW");
```

```
return;
}
for(i=q->front;irear;i++)
printf("%d\t",q->array[i]);
}
int main()
{
int max,ele,i,choice,n=0,y,z;
queue q;
stack s;
printf("\n Enter the maximum elements:");
scanf("%d",&max);
q=createqueue(max);
s=createstack(max);
while(1)
{
printf("\n Menu:1.Insert 2.Display reversed order 3.exit");
printf("\n Enter the choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("\n Enter the element:");
scanf("%d",&ele);
enqueue(q,ele);
n++;

break;

case 2:

printf("\n Contents of the queue:");

display(q);

for(i=0;i< capacity;i++)

{

z=frontanddelete(q),s;

push(z,s);

}

q->front=-1;

q->rear=-1;

for(i=0;i< capacity;i++)

{

y=topandpop(s);

enqueue(q,y);

}

printf("\n Reversed contents are:");

display(q);

break;

case 3:

exit(0);

}

}

}
```

Q5)WAP to reverse a queue using another Queue.

Example: Input: {1, 2, 3, 4, 5} Output: 5 4 3 2 1

```
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int s[MAX];
    int front,rear;
}st;
stack <int> stack1;

/* Function to check if the queue is full */
int full()
{
    if(st.rear >= MAX - 1)
        return 1;
    else
        return 0;
}

/* Function to check if the queue is empty */
int empty()
{
    if(st.front == -1)
        return 1;
    else
        return 0;
}

/* Function to insert elements in a queue */
void enqueue(int num)
{
    if(st.front == -1)
        st.front++;
    st.rear++;
    st.s[st.rear] = num;
}

/* Function to delete elements from the queue */
int dequeue()
{
    int x;
    x = st.s[st.front];
    if(st.front==st.rear)
        st.front=st.rear=-1;
    else
        st.front++;
    return x;
```

```
}

/* Function to display queue elements */
void display()
{
    int i;
    if(empty())
        printf("\nEMPTY QUEUE\n");
    else
    {
        printf("\nQUEUE ELEMENTS : ");
        for(i = st.front ; i <= st.rear ; i++)
            printf("%d ",st.s[i]);
    }
    printf("\n");
}

/* Function to reverse a queue using a stack */
void reverse_queue_using_stack()
{
    while(!(st.front == st.rear))
    {stack1.push(dequeue());}
    stack1.push(dequeue());
    printf("\nREVERSED QUEUE : ");
    while(!stack1.empty())
    {
        printf("%d ",stack1.top());   // Print the top element of the stack
        stack1.pop();
    }
    printf("\n");
    exit(0);
}

/* Main function */
int main()
{
    int num,choice;
    st.front = st.rear = -1;
    printf("\nREVERSING A QUEUE USING STACKS\n");
    printf("\n1.ENQUEUE\n2.DEQUEUE\n3.DISPLAY\n4.REVERSE\n5.EXIT\n");
    while(1)
    {
        printf("\nEnter the choice : ");
        scanf("%d",&choice);
        switch (choice)
        {
            case 1:
                if(full())
                {
                    printf("\nQUEUE IS FULL\n");
                }
                else
                {
                    printf("\nEnter data : ");
                    scanf("%d",&num);
                    enqueue(num);
                }
                break;
            case 2:
                if (empty())
                {
                    printf("\nEMPTY QUEUE\n");
                }
                else
                    printf("\nDequeued Element :
%d",dequeue());
                break;
            case 3:
                display();
                break;
            case 4: reverse_queue_using_stack();
                break;
            default: exit(0);
        }}
    return 0;
```

```
}
```

Q6)6. WAP to interleave the first half of the queue with second half.

Examples: Input: 1 2 3 4 Output: 1 3 2 4

Input: 11 12 13 14 15 16 17 18 19 20 Output: 11 16 12 17 13 18 14 19 15 20

```
class Cell
{
friend class Queue;
public:
    Cell(void *ptr, Cell *lst)
    {
        item = ptr;
        next = lst;
        copyof = NULL;
    }
    Cell(void *ptr, Cell *lst, void *(* cpfn)(void *))
    {
        item = ptr;
        next = lst;
        copyof = cpfn;
    }
    void *copyon() { return copyof(item); }
private:
    void *item;

    Cell *next;

    void *(* copyof)(void *);

};


class Queue

{

public:

    Queue(void (* d)(void *)) { dispfn = d; head = NULL; tail = NULL; }

    Queue() { dispfn = intdisplay; cpfn = copyof; head = NULL; tail = NULL; }

    void  enqueue(void *t)

    {

        Cell *ptr;

        if (t == NULL) return;

        Cell *h = new Cell(t, NULL, cpfn);

        if (head == NULL)

            head = h;

        else

            tail->next = h;

        tail = h;

    }

    void *dequeue()

    {

        if (head == NULL) return NULL;
```

```
      void *ptr = head;
      void *t = head->item;
      head = head->next;
      delete ptr;
      if (head == NULL) tail = NULL;
      return t;
   }
   Queue *mult(int n);
   Cell  *header() { return head; }
   Queue *merge(Queue *q);
   void display()
   {
     if (head == NULL) { printf("(empty)\n");  return; }
     for (Cell *t=head ; t != NULL ; t=t->next) (dispfn)(t->item);
     printf("\n");
   }
   int   empty()  {  return head == NULL;  }
private:
   Cell *head;
   Cell *tail;
   void (* dispfn)(void *);
   void *(* cpfn)(void *);
};


int cmpfunc(void *a, void *b)
```

```
{
    if (*(int *)a < *(int *)b) return -1;
    else
    if (*(int *)a > *(int *)b) return 1;
    else
        return 0;
}


Queue *Queue::mult(int n)
{
    for (Cell *h = head ; h != NULL ; h = h->next) *(int *)h->item *= n;
    return this;
}


Queue *Queue::merge(Queue *q)
{
    Cell *r = q->header();
    Cell *s = head;
    Cell *h = NULL;
    Cell *p = NULL;
    while (s != NULL || r != NULL)
    {
        if ((s == NULL && r != NULL) ||
             (s != NULL && r != NULL && cmpfunc(r->item,s->item) < 0))
        {
          void *t = r->copyon();
          if (h == NULL) h = p = new Cell(t, NULL, r->copyof);
          else
          {
            p->next = new Cell(t, NULL, r->copyof);
            p = p->next;
          }
          r = r->next;
        }
        else
        if ((r == NULL && s != NULL) ||
             (s != NULL && r != NULL && cmpfunc(r->item,s->item) >= 0))
        {
          if (h == NULL) h = p = s;
          else
          {
            p->next = s;
            p = p->next;
          }
          s = s->next;
        }
    }
    head = h;
```

```
    tail = p;
    return this;
}

int main()
{
    Queue *s = new Queue();
    Queue *r = new Queue();
    s->enqueue(new int(4));
    s->enqueue(new int(9));
    s->enqueue(new int(13));
    s->enqueue(new int(16));
    s->enqueue(new int(21));
    s->display();
    r->enqueue(new int(1));
    r->enqueue(new int(5));
    r->enqueue(new int(17));
    r->enqueue(new int(18));
    r->enqueue(new int(25));
    r->display();
    s->merge(r);
    s->display();
    r->display();
    s->mult(6);
    s->display();
```

```
   return 1;
}
```

Q7)Suppose there is a circle. There are n petrol pumps on that circle. You are given two

sets of data;

a. The amount of petrol that every petrol pump has.

b. Distance from that petrol pump to the next petrol pump.

Calculate the first point from where a truck will be able to complete the circle (The

truck will stop at each petrol pump and it has infinite capacity). Assume for 1-litre

petrol, the truck can go 1 unit of distance.

Example, let there be 4 petrol pumps with amount of petrol and distance to next

petrol pump value pairs as {4, 8}, {6, 5}, {7, 3} and {4, 5}.

Output: The first point from where the truck can make a circular tour is 2nd petrol

pump. Therefore, answer should be “start = 1” (index of 2nd petrol pump).

```
#include<stdio.h>

struct circle

{

    int dis,pet;

};

int main()

{

    int n;

    printf("Enter the no. of petrol pumps:");

    scanf("%d",&n);

    struct circle st[n];

    for(int i=0;i<n;i++)

    {

        printf("Enter the amount of petrol that every petrol pump%d has:",i+1);
        scanf("%d",&st[i].pet);

        printf("Enter distance from petrol pump%d to petrol pump%d:",i+1,i+2);

        scanf("%d",&st[i].dis);

    }

    int p,f;

    for(int i=0;i<n;i++)

    {

        p=0;

        f=1;

        for(int j=i;j<n && f==1;j++)

        {

            p+=st[j].pet;

            if(p<st[j].dis)

                f=0;

            p-=st[j].dis;

        }

        for(int j=0;j<i && f==1;j++)

        {

            p+=st[j].pet;

            if(p<st[j].dis)

                f=0;

            p-=st[j].dis;

        }

        if(f==1)

        {
            printf("Start=%d",i);
            break;
        }
    }
    return 0;
}
```