DS WEEK-7

PGNO-117:

```
#include <stdio.h>
#define MAX_SIZE 100
int stack[MAX_SIZE];
int top = -1;
void push(int element) {
    if (top >= MAX_SIZE - 1) {
        printf("Stack overflow\n");
        return;
    }
    stack[++top] = element;
}
int pop() {
    if (top < 0) {
        printf("Stack underflow\n");
        return -1;
    }
    return stack[top--];
}
int main() {
    push(10);
    push(20);
    push(30);
    printf("%d\n", pop());
    printf("%d\n", pop());

    printf("%d\n", pop());

    printf("%d\n", pop());

    push(40);

    printf("%d\n", pop());

    return 0;

}
```

PGNO-118:

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int bogie_number;
    struct Node* next;
};

struct Train {
    struct Node* head;
};

struct Train* create_train() {
    struct Train* train = (struct Train*)malloc(sizeof(struct Train));
    train->head = NULL;
    return train;
}
void insert_bogie(struct Train* train, int bogie_number) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->bogie_number = bogie_number;
    new_node->next = NULL;

    if (train->head == NULL) {
        train->head = new_node;
    } else {
        struct Node* curr = train->head;
        while (curr->next != NULL) {
            curr = curr->next;
        }
        curr->next = new_node;
    }

    printf("Bogie %d inserted.\n", bogie_number);
}
void delete_bogie(struct Train* train, int bogie_number) {
    if (train->head == NULL) {
        printf("Train is empty.\n");
        return;
    }

    if (train->head->bogie_number == bogie_number) {
        struct Node* temp = train->head;
        train->head = train->head->next;
        free(temp);
        printf("Bogie %d deleted.\n", bogie_number);
        return;
    }

    struct Node* curr = train->head;
    while (curr->next != NULL && curr->next->bogie_number !=
bogie_number) {
        curr = curr->next;
    }

    if (curr->next == NULL) {
        printf("Bogie %d not found.\n", bogie_number);
    } else {
        struct Node* temp = curr->next;
        curr->next = curr->next->next;
        free(temp);
        printf("Bogie %d deleted.\n", bogie_number);
    }
}
void search_bogie(struct Train* train, int bogie_number) {
    if (train->head == NULL) {
        printf("Train is empty.\n");
        return;
    }

    struct Node* curr = train->head;
    while (curr != NULL && curr->bogie_number != bogie_number) {
        curr = curr->next;
    }

    if (curr == NULL) {
        printf("Bogie %d not found.\n", bogie_number);
    } else {
        printf("Bogie %d found.\n", bogie_number);
    }
}
void print_train(struct Train* train) {
    if (train->head == NULL) {
        printf("Train is empty.\n");
        return;
    }

    printf("Train: ");
    struct Node* curr = train->head;
    while (curr != NULL) {
        printf("%d ", curr->bogie_number);
        curr = curr->next;
    }
    printf("\n");
}
int main() {
    struct Train* train = create_train();
    int choice, bogie_number;

    do {
printf("\nTrain Operations\n");
printf("1. Insert Bogie\n");
printf("2. Delete Bogie\n");
printf("3. Search Bogie\n");
printf("4. Print Train\n");
printf("0. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 0:
        printf("Exiting program.\n");
        break;
    case 1:
        printf("Enter bogie number: ");
        scanf("%d", &bogie_number);
        insert_bogie(train, bogie_number);
        break;
    case 2:
        printf("Enter bogie number: ");
        scanf("%d", &bogie_number);
        delete_bogie(train, bogie_number);
        break;
    case 3:
        printf("Enter bogie number: ");
        scanf("%d", &bogie_number);
        search_bogie(train, bogie_number);
        break;
    case 4:
        print_train(train);
        break;
    default:
        printf("Invalid choice.\n");
        break;
}
} while (choice != 0);

return 0;
}
```

OUTPUT:

```
Train Operations
1. Insert Bogie
2. Delete Bogie
3. Search Bogie
4. Print Train
0. Exit
Enter your choice: 1
Enter bogie number: 6
Bogie 6 inserted.

Train Operations
1. Insert Bogie
2. Delete Bogie
3. Search Bogie
4. Print Train
0. Exit
```

```
Enter your choice: 2
Enter bogie number: 6
Bogie 6 deleted.
```

PGNO119:

https://www.hackerearth.com/practice/data-structures/trees/heapspriority-queues/practice-problems/algorithm/queues-content-problem/

```
#include<stdio.h>
int main()
{
  int a[1000000];
  int n,i,max=0;
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
    scanf("%d",&a[i]);
    if(a[i]>max)
    {
      max=a[i];
    }
  }
  int t;
  scanf("%d",&t);
  while(t--)
  {
    int c,k;
    scanf("%d",&c);
    if(c==1)
    {
      scanf("%d",&k);
      if(k>max)
      {
        max=k;
      }
      a[n]=k;
      n++;
    }
    else if(c==2)
    {
      printf("%d\n",max);
```

```
        }
    }
}
```

PGNO120:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_CUSTOMERS 100

typedef struct {
    char name[50];
    int tickets;
} Customer;

Customer queue[MAX_CUSTOMERS];
int front = 0;
int rear = -1;
int itemCount = 0;

void enqueue(Customer newCustomer) {
    if(itemCount == MAX_CUSTOMERS) {
        printf("Queue is full, cannot add new customers.\n");
    } else {
        rear = (rear + 1) % MAX_CUSTOMERS;
        queue[rear] = newCustomer;
        itemCount++;
        printf("%s has been added to the queue.\n", newCustomer.name);
    }
}

Customer dequeue() {
    if(itemCount == 0) {
        printf("Queue is empty, cannot dequeue customers.\n");
        Customer emptyCustomer = {"", 0};
        return emptyCustomer;
    } else {
        Customer nextCustomer = queue[front];
        front = (front + 1) % MAX_CUSTOMERS;
        itemCount--;
        printf("%s has been removed from the queue.\n",
nextCustomer.name);
        return nextCustomer;
    }
}
int search(char name[], int tickets) {
    int i, j;
    for(i = front, j = 0; j < itemCount; j++, i = (i + 1) %
MAX_CUSTOMERS) {
        if(strcmp(queue[i].name, name) == 0 && queue[i].tickets == tickets)
{
            return 1; // customer found in the queue
        }
    }
    return 0; // customer not found in the queue
}

int main() {
    int choice, numCustomers = 0;
    char name[50];
    int tickets;
       do {
        printf("\nEnter your choice:\n");
        printf("1. Add new customer\n");
        printf("2. Remove next customer\n");
        printf("3. Search for a customer\n");
        printf("4. Display queue status\n");
        printf("5. Exit\n");
        scanf("%d", &choice);

        switch(choice) {
            case 1: // Add new customer
                if(numCustomers == MAX_CUSTOMERS) {
                    printf("Maximum number of customers have already
registered.\n");
                } else {
                    printf("\nEnter customer name: ");
                    scanf("%s", name);
                    printf("Enter number of tickets: ");
                    scanf("%d", &tickets);
                    Customer newCustomer = {name, tickets};
                    enqueue(newCustomer);
                    numCustomers++;
                }
                break;

            case 2: // Remove next customer
                dequeue();
                break;

            case 3: // Search for a customer
                printf("\nEnter customer name: ");
                scanf("%s", name);
                printf("Enter number of tickets: ");
                scanf("%d", &tickets);
                if(search(name, tickets)) {
                    printf("%s with %d tickets has already registered and is in
the queue.\n", name, tickets);
                } else {
                    printf("%s with %d tickets has not registered.\n", name,
tickets);
                }
                break;

     case 4: // Display queue status
if(itemCount == 0) {
printf("The queue is currently empty.\n");
} else {
printf("The queue currently has %d customer(s) waiting:\n", itemCount);
int i, j;
for(i = front, j = 0; j < itemCount; j++, i = (i + 1) % MAX_CUSTOMERS) {
printf("%d. %s with %d ticket(s)\n", j+1, queue[i].name, queue[i].tickets);
}
}
break;
        case 5: // Exit
            printf("Exiting the program...\n");
            break;
        default: // Invalid choice
            printf("Invalid choice, please try again.\n");
            break;
    }

} while(choice != 5);

return 0;
}
```

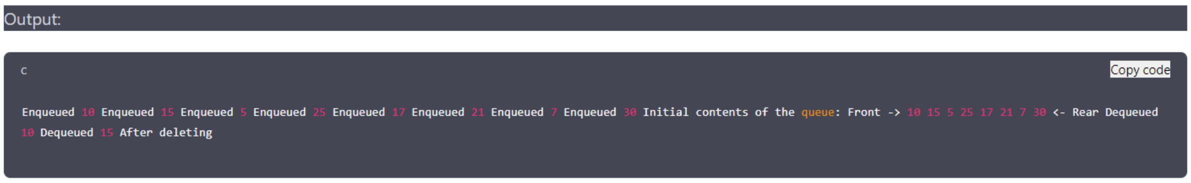
PGNO-121:

```
#include <stdio.h>
#define MAX_SIZE 5

int queue[MAX_SIZE];
int front = -1, rear = -1;
void insert(int value) {
    // Check if queue is full
    if ((front == 0 && rear == MAX_SIZE - 1) || (rear == front - 1)) {
        printf("Queue Overflow\n");
    }   else if (front == -1) {
        front = 0;
        rear = 0;
        queue[rear] = value;
    }   else if (rear == MAX_SIZE - 1 && front != 0) {
        rear = 0;
        queue[rear] = value;
    }
    else {
        rear++;
        queue[rear] = value;
    }
}

void delete() {
    // Check if queue is empty
    if (front == -1) {
        printf("Queue Underflow\n");
    }   else if (front == rear) {
        printf("Deleted element: %d\n", queue[front]);
        front = -1;
        rear = -1;
    }   else if (front == MAX_SIZE - 1) {
        printf("Deleted element: %d\n", queue[front]);
        front = 0;
    }
    else {
        printf("Deleted element: %d\n", queue[front]);
        front++;
    }
}
void display() {
    // Check if queue is empty
    if (front == -1) {
        printf("NULL\n");
    }
    else {
        int i;
        printf("Elements in the queue: ");
        if (rear >= front) {
            for (i = front; i <= rear; i++)
                printf("%d ", queue[i]);
        }
        else {
            for (i = front; i < MAX_SIZE; i++)
                printf("%d ", queue[i]);
            for (i = 0; i <= rear; i++)
                printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice, value;
    do {
        printf("Enter your choice:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to be inserted: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 4);
    return 0;
}
```

PGNO-123:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 10

// Queue data structure using array
typedef struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
} Queue;
```

```
// Function to create a new empty queue
Queue* createQueue() {
    Queue* q = (Queue*) malloc(sizeof(Queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

// Function to check if queue is full
int isFull(Queue* q) {
    if (q->rear == MAX_SIZE - 1) {
        return 1;
    }
    return 0;
}

// Function to check if queue is empty
int isEmpty(Queue* q) {
    if (q->front == -1 && q->rear == -1) {
        return 1;
    }
    return 0;
}

// Function to add an item to the queue
void enqueue(Queue* q, int item) {
    if (isFull(q)) {
        printf("Queue is full.\n");
    } else {
        if (q->front == -1) {
            q->front = 0;
        }
        q->rear++;
        q->items[q->rear] = item;
        printf("Enqueued %d\n", item);
    }
}

// Function to remove an item from the queue
int dequeue(Queue* q) {
    int item;
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        item = -1;
    } else {
        item = q->items[q->front];
        q->front++;
        if (q->front > q->rear) {
            q->front = q->rear = -1;
        }
    }
    return item;
}

// Function to display the contents of the queue
void display(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
    } else {
        printf("Front -> ");
        for (int i = q->front; i <= q->rear; i++) {
            printf("%d ", q->items[i]);
        }
        printf(" <- Rear\n");
    }
}

int main() {
    Queue* q = createQueue();

    // Inserting elements
    enqueue(q, 10);
    enqueue(q, 15);
    enqueue(q, 5);
    enqueue(q, 25);
    enqueue(q, 17);
    enqueue(q, 21);
    enqueue(q, 7);
    enqueue(q, 30);
    // Displaying the initial contents of the queue
    printf("Initial contents of the queue:\n");
    display(q);

    // Deleting two elements
    dequeue(q);
    dequeue(q);

    // Displaying the contents of the queue after deleting two elements
    printf("\nAfter deleting two elements:\n");
    display(q);

    // Inserting 37 and 20
    enqueue(q, 37);
    enqueue(q, 20);

    // Displaying the contents of the queue after inserting 37 and 20
    printf("\nAfter inserting 37 and 20:\n");
    display(q);

    return 0;
}
```



```
Enqueued 10 Enqueued 15 Enqueued 5 Enqueued 25 Enqueued 17 Enqueued 21 Enqueued 7 Enqueued 30 Initial contents of the queue: Front -> 10 15 5 25 17 21 7 30 <- Rear Dequeued
10 Dequeued 15 After deleting
```

PGNO-124:

```
#include <stdio.h>
#include <stdlib.h>

struct sNode {
int data;
struct sNode* next;
};
```

```
void push(struct sNode** top_ref, int new_data);

int pop(struct sNode** top_ref);

struct queue {
struct sNode* stack1;
struct sNode* stack2;
};

void enQueue(struct queue* q, int x)
{
push(&q->stack1, x);
}

int deQueue(struct queue* q)
{
int x;

if (q->stack1 == NULL && q->stack2 == NULL) {
printf("Q is empty");
getchar();
exit(0);
}

if (q->stack2 == NULL) {
while (q->stack1 != NULL) {
x = pop(&q->stack1);
push(&q->stack2, x);
}
}

x = pop(&q->stack2);
return x;
}

void push(struct sNode** top_ref, int new_data)
{
struct sNode* new_node = (struct sNode*)malloc(sizeof(struct sNode));
if (new_node == NULL) {
printf("Stack overflow \n");
getchar();
exit(0);
}

new_node->data = new_data;

new_node->next = (*top_ref);

(*top_ref) = new_node;
}

int pop(struct sNode** top_ref)
{
int res;
struct sNode* top;

if (*top_ref == NULL) {
printf("Stack underflow \n");
getchar();
exit(0);
}
else {
top = *top_ref;
res = top->data;
*top_ref = top->next;
free(top);
return res;
}
}

int main()
{
struct queue* q = (struct queue*)malloc(sizeof(struct queue));
q->stack1 = NULL;
q->stack2 = NULL;
enQueue(q, 1);
enQueue(q, 2);
enQueue(q, 3);
printf("%d ", deQueue(q));
printf("%d ", deQueue(q));
printf("%d ", deQueue(q));

return 0;
}
```

PGNO-125:

https://www.codechef.com/problems/CHFQUEUE/

```
#include <stdio.h>

#include <stdbool.h>

#define MOD 1000000007
typedef long long int lli;
inline int input(){
int a=0; bool flag = false;
char c;
c=getchar_unlocked();
while(c<33){
c=getchar_unlocked();
}
if(c == 45){
flag = true;
c=getchar_unlocked();
}
while(c>=33){
a=(a<<3)+(a<<1)+(c-'0');
c=getchar_unlocked();
}
if(flag){
a *= -1;
}

return a;

};
int stack[1000002], pos[1000002], top1 = -1, top2 = -1;
void push_stack(int val){
stack[++top1] = val;
}
```

```
void push_pos(int val){
pos[++top2] = val;
}
void pop_stack(){
top1--;
}
void pop_pos(){
top2--;
}
int main(void) {
int n, k, arr[1000002], i, j;
lli f, total_fear = 1;
n = input();
k = input();
for(i=1; i<=n; i++){
arr[i] = input();
}
for(i=n; i>=1; i--){
if(top1 == -1){
push_stack(arr[i]);
push_pos(i);
}
else{
while(stack[top1] >= arr[i]){
pop_stack();
pop_pos();
}
if(top1 > -1){
f = pos[top2] - i + 1;
total_fear = (total_fear * f) % MOD;
}
push_stack(arr[i]);
push_pos(i);
}
}
printf("%lli\n", total_fear);
return 0;
}
```

PGNO-126:

https://www.codechef.com/problems/CAC202?tab=statement

```
#include <stdio.h>
int main(){
    int first;
    scanf("%d",&first);
    while(first--){
        int v,x,m=0;
   scanf("%d",&v);
        int s[v],a[v],top=-1;
        for(int i=0;i<v;i++){
            a[i]=0;
        }
        while(v--){
            scanf("%d",&x);
            if(x>m){
                printf("%d ",x);
                for(int i=m+1;i<x;i++){
                    top++;
                    s[top]=i;
                }
                m=x;
            }
            else{
                a[x]=1;
                while(top!=-1&&a[s[top]]==1){
                    a[s[top]]=0;
                    printf("%d ",s[top]);
                    top--;
                }
            }
        }
        while(top!=-1){
            printf("%d",s[top]);
            top--;
        }
        printf("\n");
    }
return 0;
}
```

PGNO-127:

https://www.hackerearth.com/problem/algorithm/queue-problem-jatinj-1addbbb7/

```
#include <stdio.h>
long long int fact(long long int n){
if(n<2)return n;
else return n*fact(n-1);
}
int main(){
long long int b1, g1;
scanf("%lld %lld", &b1, &g1);

   int arrangements=0;
   if(g1>=b1)printf("0");
   else if(b1<3)printf("%d", b1);
   else {
   int b=11;int g=11;
   int dp[b+1][g+1];
   for(int i=0;i<=b;i++){
   for(int j=0;j<=g;j++){
   if(j==0)dp[i][j]=1;
   else dp[i][j]=0;
   }
   }
   dp[0][0]=0;
   for(int i=1;i<=b;i++){
   for(int j=1;j<i;j++){
   dp[i][j]=dp[i-1][j]+dp[i][j-1];
   }
   }
   long long int ans = 1;
   ans = ans*dp[b1][g1]*fact(b1)*fact(g1);
   printf("%lld",ans);//*fact(b)*fact(g));
   }
   }
```

PGNO-128:

https://www.hackerearth.com/practice/basic-programming/implementation/basics-of-implementation/practice-problems/algorithm/aniruddhas-queue-4/

```
#include<stdio.h>
#define ll long long int
int main()
{
int c;
scanf("%d",&c);
while(c--)
{
int n;
scanf("%d",&n);
ll ar[n],cnt=0;
for(int i=0;i<n;++i)
{
scanf("%lld",&ar[i]);
cnt+=ar[i];
}
ll m;
scanf("%lld",&m);
m%=cnt;
int ans;
if(m==0)
{
for(int i=n-1;i>=0;--i)
{
if(ar[i]!=0)
{
ans=i;
break;
}
}
}
else
{
for(int i=0;i<n;++i)
{
m-=ar[i];
if(m<=0)
{
ans=i;
break;
}

}
}
printf("%d\n",ans+1);
}
return 0;
}
```

PGNO-129:

https://www.hackerrank.com/challenges/queue-using-two-stacks/problem?isFullscreen=true

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

struct stackNode {
    int data;
    struct stackNode *next;
};

void push(struct stackNode **stack, int data){
    struct stackNode *newNode = malloc(sizeof(struct stack
Node));
    newNode->data = data;
    newNode->next = *(stack);
    *(stack) = newNode;
}

int peek(struct stackNode **stack){
    if (*(stack) == NULL){
        return -1;
    }else return (*(stack))->data;
}


int pop(struct stackNode **stack){
    int outputValue=(*(stack))->data;
    *(stack) = (*(stack))->next;
    return outputValue;
}

void transferStacks(struct stackNode **stack1, struct stac
kNode **stack2){
    int tmpNum;
    while ((*stack1)!=NULL){
        tmpNum = pop(stack1);
        push(stack2,tmpNum);
    }
}

int main() {
    int numLineToRead, comNum, inputNum;

    struct stackNode *stackTmp1 = NULL, *stackTmp2 = NULL,
 *stackTmp3 = NULL, *stackTmp4 = NULL;
    struct stackNode **stack1 = &stackTmp1, **stack2 = &st
ackTmp2, **stack3 = &stackTmp3, **stack4 = &stackTmp4;

    scanf("%d",&numLineToRead);
    for (int k = 0;k<numLineToRead;k++){
        scanf("%d",&comNum);
        if (comNum==1){
            scanf("%d",&inputNum);
            push(stack1, inputNum);
        }else if (comNum == 2){
            push(stack3, 2);
        }else if (comNum == 3){
            push(stack3, 3);
        }else{
            printf("Wrong Input Format.");
        }
    }

    transferStacks(stack1,stack2);
    transferStacks(stack3,stack4);

    while((*stack4)!=NULL){
        if (pop(stack4) == 2){
            pop(stack2);
        }else{
            printf("%d\n",peek(stack2));
        }
    }
    return 0;
}
```

PGNO-130:

https://www.hackerearth.com/practice/data-structures/arrays/1-d/practice-problems/algorithm/long-atm-queue-3/

```
#include <stdio.h>
int main(){
int num,i;
scanf("%d", &num);
int a[num];
for(i=0;i<num;i++)
scanf("%d",&a[i]);
int cnt=1;
for(i=1;i<num;i++){
if(a[i]>=a[i-1])
continue;
else
cnt++;
} // Reading input from STDIN
printf("%d\n",cnt);
return 0; // Writing output to STDOUT
}
```