NAME : BHANU PRATAP

Admission No-IITP00529

Date:8thMarch23

Q1. Implementation of fixed size STACK data structure in C using array. The stack should have push, pop, create, isempty, isfull functions

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE_MAX 20
struct Stack {
  int items[SIZE_MAX];
  int top;
};
struct Stack * createStack();
void push(struct Stack * stack, int item);
int pop(struct Stack * stack);
int is_Empty(struct Stack * stack);
int is_Full(struct Stack * stack);
struct Stack * createStack() {
  struct Stack * stack = (struct Stack * ) malloc(sizeof(struct Stack));
  stack -> top = -1;
  return stack;
}
void pushElement(struct Stack * stack, int item) {
  if (is_Full(stack)) {
    printf("Stack is full\n");
```

```
  } else {
    stack -> top++;
    stack -> items[stack -> top] = item;
    printf("Element is pushed in %d\n", item);
  }
}
int popNode(struct Stack * stack) {
  int item;
  if (is_Empty(stack)) {
    printf("Stack is empty\n");
    return -1;
  } else {
    item = stack -> items[stack -> top];
    stack -> top--;
    return item;
  }
}
int is_Empty(struct Stack * stack) {
  return stack -> top == -1;
}
// check if the stack is full
int is_Full(struct Stack * stack) {
  return stack -> top == SIZE_MAX - 1;
}
int main() {
```

```
 struct Stack * stack = createStack();
 pushElement(stack, 1);
 pushElement(stack, 2);
 pushElement(stack, 3);
 pushElement(stack, 4);
 pushElement(stack, 5);
 pushElement(stack, 6);
 pushElement(stack, 7);
 pushElement(stack, 8);
 pushElement(stack, 9);
 pushElement(stack, 10);
 pushElement(stack, 11);
 printf("Element popped out: %d\n", popNode(stack));
 printf("Element popped out: %d\n", popNode(stack));
 printf("Element popped out: %d\n", popNode(stack));
 printf("Element popped out: %d\n", popNode(stack));
 return 0;
}
```

Q2. Implementation of dynamic size STACK data structure in C using linked list. The stack should have push, pop, create, isempty, isfull functions.

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

struct stack{
   int size ;
```

```
    int top;
    int * arr;
};

int isEmpty(struct stack* ptr){
    if(ptr->top == -1){
        return 1;
      }
      else{
        return 0;
      }
}

int isFull(struct stack* ptr){
    if(ptr->top == ptr->size - 1){
      return 1;
    }
    else{
      return 0;
    }
}

void push(struct stack* ptr, int val){
    if(isFull(ptr)){
      printf("Stack Overflow! Cannot push %d to the stack\n", val);
```

```
    }
    else{
        ptr->top++;
        ptr->arr[ptr->top] = val;
    }
}

int pop(struct stack* ptr){
    if(isEmpty(ptr)){
        printf("Stack Underflow! Cannot pop from the stack\n");
        return -1;
    }
    else{
        int val = ptr->arr[ptr->top];
        ptr->top--;
        return val;
    }
}

int main(){
    struct stack *sp = (struct stack *) malloc(sizeof(struct stack));
    sp->size = 10;
    sp->top = -1;
    sp->arr = (int *) malloc(sp->size * sizeof(int));
    printf("Stack has been created successfully\n");
```

```
    printf("Before pushing, Full: %d\n", isFull(sp));
    printf("Before pushing, Empty: %d\n", isEmpty(sp));
    push(sp, 1);
    push(sp, 65);
    push(sp, 39);
    push(sp, 75);
    push(sp, 3);
    push(sp, 4);
    push(sp, 67);
    push(sp, 42);
    push(sp, 55);
    push(sp, 26); // ---> Pushed 10 values
    push(sp, 46); // Stack Overflow since the size of the stack is 10
    printf("After pushing, Full: %d\n", isFull(sp));
    printf("After pushing, Empty: %d\n", isEmpty(sp));
     printf("Popped %d from the stack\n", pop(sp)); // --> Last in first out!
    printf("Popped %d from the stack\n", pop(sp)); // --> Last in first out!
    printf("Popped %d from the stack\n", pop(sp)); // --> Last in first out!
    return 0;
}
```

Q3. Implementation of QUEUE data structure in C using linked list. It should have enqueue, dequeue, isempty and size functions

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

```
typedef struct Node {
int data;
struct Node* next;
} Node;
typedef struct Queue {
Node* front;
Node* rear;
int size;
} Queue;
Node* newNode(int data) {
Node* temp = (Node*)malloc(sizeof(Node));
temp->data = data;
temp->next = NULL;
return temp;
}
Queue* createQueue() {
Queue* queue = (Queue*)malloc(sizeof(Queue));
queue->front = queue->rear = NULL;
queue->size = 0;
return queue;
}
int isEmpty(Queue* queue) {
return (queue->front == NULL);
}
void enqueue(Queue* queue, int data) {
```

```
Node* temp = newNode(data);
queue->size++;
if (queue->rear == NULL) {
queue->front = queue->rear = temp;
return;
}
queue->rear->next = temp;
queue->rear = temp;
}
int dequeue(Queue* queue) {
if (isEmpty(queue)) {
return -1;
}
Node* temp = queue->front;
int data = temp->data;
queue->front = queue->front->next;
if (queue->front == NULL) {
queue->rear = NULL;
}
free(temp);
queue->size--;
return data;
}
int size(Queue* queue) {
return queue->size;
```

```
}

int main() {

Queue* queue = createQueue();

enqueue(queue, 11);

enqueue(queue, 12);

enqueue(queue, 13);

enqueue(queue, 14);

enqueue(queue, 15);

enqueue(queue, 16);

printf("Dequeued Element: %d\n", dequeue(queue));

printf("Dequeued Element: %d\n", dequeue(queue));

printf("Size of Queue: %d\n", size(queue));

return 0;

}
```