29/01/20

1. Stack Implementation using single linked list

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
 int data;
 struct Node *next;
};
struct Stack{
   struct Node *top;
 };
struct Node* createNode(int data){
   struct Node* newNode = (struct
   Node*) malloc(sizeof(struct Node));
      newNode->data = data;
      newNode->next = NULL;
      return newNode;
}
struct Stack* createStack(){
  struct Stack *stack = (struct
Stack*) malloc(sizeof(struct Stack));
    stack->top = NULL;
    return stack;
}
int isEmpty(struct Stack* stack){
    return stack->top == NULL;
  }
void push(struct Stack* stack, int data){
 struct Node* newNode = createNode(data);
   newNode->next = stack->top;
   stack->top = newNode;
}
int pop(struct Stack* stack){
    if(isEmpty(stack)){
```

```
pr...
return -1;
}
struct Node* poppedNode = stack->top;
int poppedData = poppedNode->data;
stack->top = poppedNode->next;
free(poppedNode);
return poppedData;
}
int peek(struct Stack* stack){
  return isEmpty(stack) ? -1 :
    stack->top->data;
}
void display(struct Stack* stack)
{ struct Node* current = stack->top;
  while(current != NULL){
    printf("%d", current->data);
    current = current->next;
  }
  printf("\n");
}
int main(){
  struct Stack* stack = createStack();
  push(stack, 1);
  push(stack, 2);
  push(stack, 3);
  printf("Stack:");
  display(stack);
  printf("Peek:%d", peek(stack));
  printf("Pop:%d\n", pop(stack));
  printf("Stack after pop:");
  display(stack);
  return 0;
}
```

SLL - Stacks, Queues, - Rev, sort, concate

Ext 29/11/24.

Output:

Enter 1. Push 2. Pop 3. -1 to stop
: Enter operation.
: 1. Enter element to push : 23
Stack elements are : 23
Enter operation : 1
Enter Element to push : 45
Stack Elements are : 45 23
Enter operation : 2
popped Element : 45
Stack Elements are : 23

2. Queue implementation using single linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node* next;
};
struct Queue{
  struct Node *front;
  struct Node *rear;
};
struct Node* createNode(int data){
    struct Node* newNode=(struct
    Node*)malloc(sizeof(struct Node));
        newNode->data=data;
        newNode->next=NULL;
        return newNode;
}
```

```
struct Queue* queue = (struct Queue*)malloc(sizeof
                    (struct Queue));
  queue->front = queue->rear = NULL;
  return queue;

}
int isEmpty(struct Queue* queue){return queue->front
                                    ==NULL;}

void enqueue(struct Queue* queue, int data)
  { struct Node* newNode = createNode(data);
  if(isEmpty(queue)){
     queue->front = queue->rear = NewNode;

  }else{
     queue->rear->next = newNode;
     queue->rear = NewNode;
  }
}
int dequeue(struct Queue* queue){
  if(isEmpty(queue)) {
    printf("Queue underflow");
    return -1;
  }
struct Node* dequeuedNode = queue->front;
  int dequeuedData = dequeuedNode->data;
  queue->front = dequeuedNode->next;
  free(dequeuedNode);
  return dequeuedData;
}
int front(struct Queue* queue){
    return isEmpty(queue)?-1:
  queue->front->data;
}
void displey(struct Queue* queue){
    struct Node* current = queue->front;
    while(current!=NULL){
    Printf("%d", current->data);
```

```
    current = current->next;
  }
  printf("\n");
 }
 int main()
{
 struct Queue* queue = createQueue();
  enqueue(queue, 1);
  enqueue(queue, 2);
  enqueue(queue, 3);
  printf("Queue:");
  display(queue);
printf("Front: %d\n", front(queue));
printf("Dequeue: %d\n", dequeue(queue));
printf("Queue after dequeue:");
display(queue);
return 0;
}
```

Output: