29-1-24

1) WAP to Implement Singley Linked List to simulate Stack and Queue Operations

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

typedef struct Node{
    int data;
    struct Node *next;
}Node;

void push(struct Node **headref,
                   int newdata);
void pop(struct Node **headref);
void Enqueue(int item, struct
                  Node ** head);
void Dequeue(struct Node **head);
void Printlist (Node *node);

void push(struct Node **headref,
                   int newdata)
{
  struct Node *newnode =
    (struct Node *) malloc(sizeof
                    (struct Node));
   newnode -> data = newdata;
   newnode -> next = (*headref);
   (*headref) = newnode;
}
```

```
void pop (struct Node **headref)
{
    struct Node *ptr;
    if (*headref ==NULL)
      printf("\n List is empty");
    else{
      ptr = *headref;
      *headref = ptr->next;
      free(ptr);
      printf("\n Node deleted
             from the beginning...");
    }
}

void Enqueue (int item, struct
               Node **head) {
  struct  Node  *ptr, *temp;
  ptr = (struct Node*)malloc
          (sizeof (struct Node));
  ptr -> data  = item;
  ptr -> next  = NULL;
  if  (*head == NULL) {
        *head = ptr;
        printf("\nNode inserted");
  }
  else {
    temp = *head;
    while (temp  -> next != NULL)
        temp = temp ->next;
   temp -> next= ptr;
   printf("\nNode inserted");
  }}
```

```
void Dequeue (struct Node **head){
  struct Node *ptr;
  if (*head == NULL)
      printf("\nList is empty");
  else {
      ptr = *head;
      *head = ptr->next;
      free(ptr);
      printf("\n Node deleted
           from the beginning...")
    }
}

void PrintList (Node *node){
  while (node != NULL) {
     printf("%d \n", node->data);
     node = node->next;
  }
}

int main (){
  int ch, new;
  Node *head=NULL;
  while (ch!=6){
    printf("\n Menu \n");
    printf("1. To perform push
        operation using stack \n");
  printf("2. To perform pop
        operation using stack \n");
  printf("3. To insert element
        using queue \n");
```

```
printf("4. To delete element using
                    queue \n");
printf("5. Display \n");
printf("6. Exit \n");
printf("Enter   your choice: \n");
scanf("%d", &ch);
switch (ch){
case 1:
  printf("Enter the data
   you want to push in the
   stack : \n");
  scanf("%d", &new);
  push (&head, new);
  break;
case 2:
   pop (&head);
   break;
case 3:
  printf("Enter the data
         you want to insert: \n");
  scanf("%d", &new);
  Enqueue(new, &head);
  break;
case 4:
    Dequeue(&head);
    break;
case 5:
  printf("Created linked list is \n");
  PrintList(head);
  break;
case 6: return 0;
default: printf("Invalid input \n");
}}
return 0;
}
```

Output:

```
Menu
1. To perform push operation usingstack
2. To perform pop operation using stack
3. To insert element using queue
4. To delete element using queue
5. Display
6. Exit
Enter your choice:
1
Enter the data you want to
push in the stack : 23

Menu
1. To perform push operation using
stack
2. To perform pop operation usingstack
3. To insert element using queue
4. To delete element usingqueue
5. Display
6. Exit
Enter your choice:
4

Node deleted from the beginning...
```

2) Leet code challenge

Intersection of two linked lists.

```
struct ListNode *getIntersectionNode
   (struct ListNode *headA,
              struct ListNode *headB){
   //& Find lengths of linked lists
   int lenA=0, lenB=0;
   struct ListNode *tempA=headA,
                   *tempB = headB;

  while (tempA! = NULL){
        lenA++;
        tempA = tempA->next;
  }
  while (tempB! = NULL){
       lenB++;
       tempB = tempB ->next;
  }

 // Move the pointer of the
  longer linked list forward
  int diff = lenA - lenB;
  if (diff >0){
        while(diff >0){
          headA = headA ->next;
          diff --;
     }
  }
 else {
   while (diff <0) {
       ~~do~~ headB = headB -> next;
```

```
      diff++; }}

// Traverse both linked lists to
find intersection point

while (headA != NULL && headB != NULL)
{
    if (headA == headB)
         return headA; // Found intersection
                              point
    }
   headA = headA → next;
  headB = headB → next;
}
return   NULL; // No intersection
                         found
}
```

Output:

```
Case 1:                       Case 2:
Input
8                             2
[4, 1, 8, 4, 5]               [1, 9, 1, 2, 4]
[5, 6, 1, 8, 4, 5]            [3, 2, 4]
2                             3
3                             1
Output:
Intersected   at '8'          Intersected at '2'
Case 3:
0
[2, 6, 4]
[1, 5]
3
2
No intersection
```

29/1/24