# INDEX

1BM22CS222

NAME: RISHI. J STD.: SEC.: 3E ROLL NO.: SUB.:

| S. No. | Date | Title | Page No. | Teacher's Sign / Remarks |
|---|---|---|---|---|
| 1 | 7/12 | LAB - 1 Practice Programs | | |
| 2 | 21/12 | Swapping two variables | | |
| | | Dynamic Memory Allocation | | |
| | | Stack Implementation | | |
| 3 | 28/12 | Infix to Postfix Conversion | | |
| | | Queue Implementation | | |
| | | Circular Queue | | |
| 4 | 18/01 | Implementation of Linked List | | |
| | | Leetcode - Ministack | | |
| 5 | 18/01 | Deletion in Linked List | | |
| 6 | 25/01 | Linked List Operations | | |
| | | Stack and Queues | | |
| 7 | 1/02 | Doubly Linked List | | |
| 8 | 15/02 | Binary Search Tree | | |
| | | Rotate List - Linked List | | |
| 9 | 22/02 | BFS | | |
| | | DFS | | |
| | | HackerRank | | |
| 10 | 29/02 | Hashing - Linear Probing | | |

Sri. T

Sri. T.

DS Lab -

Outputs -

1) Enter username : Rishi
Enter password : 3110
Enter amount to deposit : 10000
Account created successfully

Enter amount to withdraw : 5000
5000 withdrawed successfully !
Remaining balance : 5000

2) Enter strings : Rishi Nidhi Nithin
Sorted order : Nidhi Nithin Rishi

3) Enter 2D Array : 1 2 3 4 5 6 7 8 9
Enter key : 6
Key found at position : 1, 2

4) Enter string : "Rishi"
Enter substring : "shi"
Substring found at 2

5) Enter array : 4 6 8 7 9 1 2
Enter key : 9
Last occurance found at 4

6) Enter array : 4 3 1 8 2
Enter key : 1
Key found at 2

7) Enter array : 1 2 3 4 5 6
Key key : 3
Key found at 2

8) Enter array : 4 3 5 1 2 6
Max element is 6 Min element is 1

Lab-2 21/12/23

1) Swapping Using Pointers

```
#include <stdio.h>
#include <stdlib.h>
void swap (int *a, int *b)
  int temp = *a;
    *a = *b;
    *b = temp;
}
int main ()
{
   int a = 3, b = 4;
   printf (" Values of a and b are %d %d ", a, b);
    swap (&a, &b);
    printf (" Values after swapping is %d %d ", a, b);
    return 0;
}
```

Output - Values of a and b are 3 4
Values after swapping is 4 3

2) Memory Allocation (malloc, calloc, realloc)

```
#include <stdio.h>
#include <stdlib.h>
int size = 3
int main() {
  int* arr = malloc(size * sizeof(int));
  int* arr2 = calloc(size, sizeof(int));
  printf("arr and arr2 can store %d integers", size);
  realloc(arr, 2* size * sizeof(int));
  printf("\n Can now store %d integers in arr and %d in arr2\n",
  2* size, size);
  for(int i=0; i< size; i++){
      arr2[i]=1;
  }
  for(int i=0; i<2* size; i++){
      arr[i]=0;
  }
  printf("arr: ");
  for(int i=0; i<2* size; i++){
    printf(" %d ", arr[i]);
  }
  printf("\n arr2: ");
  for(int i=0; i<size; i++){
    printf(" %d ", arr[i]);
  }
  free(arr); free(arr2);
  return 0;
}
```

Output - arr and arr2 can store 2 integers
can now store 6 integers in arr and 3 integers in arr2
arr: 0 0 0 0 0 0
arr2: 1 1 1

3) Stack Implementation -

```
#include <stdio.h>
#include <stdlib.h>
#define size 10
int pos = -1;
int stack[size];
void push(int a);
int pop();
void display();
int main(){
  printf("1. Push In 2. Pop In 3. Display stack In 4. Exit In Enter choice:");
  int choice;
  int a;
  scanf("%d", &choice);
  while(choice != 4){
    switch(choice){
  case 1:
    printf("Enter integer to be pushed");
    scanf("%d", &a);
    push(a);
    break;
  case 2:
    a = pop();
    printf("Integer popped = %d \n", a);
    break;
  case 3:
    display();
    break;
  }
  printf("Enter choice");
  scanf("%d", &choice);
  }
}
void push(int a){
  if(pos == a){
    printf("Stack Overflow Condition");
```

```
        return ;
    }
    Stack[++pos] = a;
}
int pop() {
    if (pos == -1) {
        printf ("Stack Underflow Condition");
        return (int) NULL;
    }
    return Stack[pos--];
}
void display () {
    for (int i = 0; i < size; i++) {
        printf ("%d", Stack[i]);
    }
    printf ("\n");
}
```

Output -

1. Push
2. Pop
3. Display
4. Exit

Enter choice : 1
Enter integer to be pushed : 3
Enter choice : 1
Enter integer to be pushed : 4
Enter choice : 2
Integer popped : 4
Enter choice : 1
Enter integer to be pushed : 5
Enter choice : 3
3 5 0 0 0 0 0 0
Enter choice : 4

24/12/2023
Next week update with Explanation

Week-2

Infix to Postfix

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define size 30

char stack[size];
int top = -1;

void push(char a)
{
  stack[++top] = a;
}

char pop()
{
  return stack[top--];
}

int precedence(char a){
    if (a == '^')
       return (3);
    else if (a == '*' || a == '/')
       return (2);
     else if (a == '+' || a == "-")
       return (1);
      else
       return (0);
}

void main(){
char infix[size] = "A+(B*C-(D/E^F)*G)*H", postfix[size];
int len = strlen(infix);
int j = 0;
for (int i=0; i<len; i++)
   {  if (infix[i] == '(')
        push('(');
      else if (infix[i] == ')')
      {
```

```
while ( stack [top] != '(')
    {
     postfix [j++] = pop ();
    }
  pop();
}
 else {
    if ( isdigit ( infix [i]))
       {
         postfix [j++] = infix [i];
       }
     else
      {
       if ( precedence ( stack [top]) < precedence ( infix [i]))
           push ( infix [i]);
         else
          {
            while ( stack [top] != "(")
            {
              postfix [j++] = pop ();
             }
              push ( infix [i]);
            }
        }
      }
        while (top != -1)
        {
          postfix [j++]= pop ();
         }
          postfix [j]= '\0';
          printf ("Postfix Expression is %.s", postfix );
}
```

OUTPUT-

Enter expression: A * B + C * D - E

Postfix : AB * CD * + E -

Program - 3

Queue Implementation -

```
# include < stdio . h >
# include < conio . h >
# define Size [30]

int queue [ Size ];
int front = -1, rear = -1
void insert ( int a) {
  if ( rear == Size -1) {
      printf ( "Queue Overflow\n");
      return ;
  }
  else {
      if ( front == -1)
          front = 0;
        queue [++rear) = a;
    }
}
void delete () {
    if ( front == -1 || front > rear ){
      printf ( "Queue Empty \n");
    }
    else
      front ++;
}
void display () {
    if ( front == -1) {
     printf ( "Queue Empty \n");
     return ;
   } printf ( " Queue :");
  for ( int i = front ; i <= rear ; i++) {
      printf ( "%d", queue [i]);
   }
   printf ( " \n ----- \n ");
}
void main (){
   int choice ;
   int a;
```

```
while (1) {
  printf ("Queue operation : \n 1. Insert \n 2. Delete \n 3. Display \n Choice : ");
  scanf ("%d", & choice);
   switch ( choice)
   {
    case 1:
      printf (" Enter Element : ");
      scanf ("%d", &a);
       insert (a);
       display ();
      break;

   case 2 :
      delete ();
      display ();
      break;

   case 3:
      display ();
      break;

   }
  }
}
```

Output -

Queue operation
1. Insert
2. Delete
3. Display
Choice : 1
Enter element : 3
Queue : 3
choice : 1
Enter element : 4
Queue : 3 4
choie : 2
Queue : 4

18/11/24

4 -4 11-1-24

Circular Queue -

```
#include <stdio.h>
#include <stdlib.h>
#define Size 5

int q[Size], f=0, r=-1;
int count=0;

void enqueue(int item) {
    if (count == Size) {
        printf("\n Queue full!");
        return;
    }
    q[(++r)%Size] = item;
    count++;
}

void dequeue() {
    if (count == 0) {
        printf("\n Queue empty");
        return;
    }
    f = (f+1) % Size;
    count--;

void display() {
    if (count == 0) {
        printf("\n Queue empty");
        return;
    }
    int front = f;
    for (int i=0; i<count; i++) {
        printf("%d", q[front]);
        front = (front+1) % Size;
    } }
```

```
int main() {
    int ch, item;
    while (1) {
        printf("\n Select choice \n 1. Enqueue \n 2. Dequeue \n 3. Display : ");
        scanf("%d", &ch);
        switch(ch) {
            case 1: ("\n Enter value to insert : ");
                    scanf("%d", &item);
                    enqueue(item);
                    break;
            case 2:
                    dequeue);
                    printf("\n Item popped");
                    break;
            case 3:
                    display();
                    break;
                    default:
                    exit(0);
        }
    }
}
```

Output -

Select choice :

1. Enqueue
2. Dequeue
3. Display

choice : 1

Enter value : 3

Queue : 3

choice : 1

Enter value : 4

Queue : 3 4

NJ 18/1/24

## Singly linked list -

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
};

void insertHead(struct node **head_ref, int new_data) {
    struct node *new_node = (struct node*) malloc (sizeof(struct node));
    new_node -> data = new_data;
    new_node -> next = (*head_ref);
    (*head_ref) = new_node;

void insertMiddle (struct node * prev_node, int new_data) {
    if (prev_node == NULL) {
    printf ("The given previous node cannot be NULL\n");
    return;
    }
    struct node *new_node = (struct node*) malloc (sizeof (struct node));
    struct node *last = *head_ref;
    new_node -> data = new_data;
    new_node -> next = NULL;
    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }
    while (last -> next != NULL)
    {
        last = last -> next;
    }
    last -> next = new_node;
    return;

void display (struct node *node)
{
    printf ("Linked List");
    while (node != NULL) {
```

```
printf ("%d", node -> data);
  node = node -> next;
}
printf ("\n");

int main (){
    struct node, *head = NULL;
    int choice = 0, a, b;
    while (choice != 3)
    {
      printf (" Select an option \n 1. Insert \n 2. Display \n 3. Exit \n Choice: ");
      scanf ("%d", &choice);
    switch (choice)
    {
     case 1: printf ("Enter the value to be inserted ");
          scanf ("%d", &a);
          int choice 2;
          printf (" \n 1. Insert at Head \n. Insert at end \n 3. Insert in
            middle \n Choice ");
         scanf ("%d", &choice2);
        switch (choice2) {
          case 1: insertAtHead (&head, a);
                  display (head);
                  break;
          case 2: insertAtEnd (&head, a);
                  display (head);
                  break;

          case 3: printf ("Enter the value after which to insert ");
                  scanf ("%d", &b);
                  struct node *temp = head;
                  while (temp != NULL && temp -> data != b) {
                     temp = temp -> next;
              }
              if (temp == NULL) {
                  printf ("Element not found in the list \n");
              }
```

```
        break;
    case 2: display (head);
            break;
    default: break;
    }
}
return 0;
}
```

Output –

```
Select an option
1. Insert
2. Display
3. Exit
Choice : 1
Enter value: 1
1. Insert at Head
2. Insert at End
3. Insert at Middle
Choice : 1
Linked List : 1
```

Week-4 18-1-24

Program-5

Deletion in Linked List

Executed
NLD
18/1/24

```
#include <stdio.h>
#include <stdlib.h>
void pop();
void end. delete();
void delete_at_pos();
void display();
void append();
struct node
{
  int data;
  struct node *next;
};
struct node *head = NULL;
void main()
{
  printf("Insert the elements in list \n");
  append();
  printf("\n 1. Delete from the begining \n 2. Delete at the end
        \n 3. Delete at a particular position \n 4. Display \n 5. Exit \n");
  int ch;
  while (ch! = 5)
  {
    printf("Enter choice :");
    scanf("%d", &ch);
    switch (ch)
    {
      case 1: pop();
              break;
```

```
case 2:
    end_delete();
    break;
case 3:
    delete_at_pos();
    break;
case 4:
    display();
    break;
default: printf("Invalid Choice");
        break;
}
}
}
void append()
{
int data, n;
printf(" Enter no. of nodes ");
scanf("%d", &n);
for (int i=0; i<n; i++)
{
struct node *last = head;
struct node *new_node;
new_node = (struct node*) malloc (sizeof (struct node));
printf ("Enter the data");
scanf("%d", &data);
new_node -> data = data;
new_node -> next = NULL;
if (head == NULL)
head = new_node;
else
{
while (last -> next != NULL)
{
  last = last -> next;
}
last -> next = new_node;
}
}
}
```

```
void beg()
{
  struct node *ptr;
   if (head == NULL)
      printf("List is empty\n");
   else
   {
    ptr = head;
    head = ptr -> next;
    free(ptr);
    printf("Node deleted from beginning\n");
   }
}
void end_delete()
{
   struct node *ptr;
   struct node *ptr1;
   if (head == NULL)
     printf("List is empty");
   else if (head -> next == NULL)
   {
    free(head);
    head = NULL;
   }
   else
   {
     ptr = head;
     ptr1 = head;
    while (ptr -> next != NULL)
    {
      ptr1 = ptr
      ptr = ptr -> next;
    }
     ptr1 -> next = NULL
     free(ptr);
     printf("Node deleted from end");
    }
  }
 void display()
 {
  struct node *p = head;
```

Leetcode Program - 1

Minstack

Solved
NJ
18/1/24

```
typedef struct {
    int array[30000];
    int min[30000];
    int top1;
    int top2;
} MinStack;

MinStack* minStackCreate() {
    MinStack* obj = (MinStack*) malloc(sizeof(MinStack));
    obj->top1 = -1;
    obj->top2 = -1;
    return obj;
}

void minStackPush(MinStack* obj, int val) {
    obj->array[++obj->top1] = val;
    if (obj->top2 == -1) {
        obj->min[++obj->top2] = val;
        return;
    }
    int mintop = obj->min[obj->top2];
    if (mintop > val) {
        obj->min[++obj->top2] = val;
        return;
    }
    else {
        obj->min[++obj->top2] = mintop;
    }
}
```

```
void minStackPop(MinStack* obj) {
    obj->top1--;
    obj->top2--;
}

int minStackTop(MinStack* obj) {
    return obj->array[obj->top1];
}

int minStackGetMin(MinStack* obj) {
    return obj->array[obj->top1];
}

void minStackFree(MinStack* obj) {
    free(obj);
}
```

Week - 6 25-1-24

Lab - 6

Program - 1

Implementing sorting, reverse and concat in a linked list -

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

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

void append (struct node **head, int new_data)
{
  struct node *new_node = (struct node *) malloc (sizeof (struct node));
  new_node -> data = new_data;
  new_node -> next = NULL;
  struct node *last = *head;
    if (*head == NULL)
        *head = new_node;
    else
    {
      while (last -> next != NULL)
        last = last -> next;
        last -> next = new_node;
    }
}

void display (struct node *head)
{
  if (head == NULL)
  {
    printf (" Linked list empty ");
    return;
  }
  printf (" Linked List ");
  while (head != NULL)
    {
```

Exit
NL
25/1/24

```
        printf ("%d", head -> data);
        head = head -> next;
      }
      printf ("\n");
  }

void bubble_sort (struct node *head)
  {
    struct node *prev;
    struct node *cur;
    int next;
    int flag = 1;
    int flag2 = 1;
    while (flag)
    {
      prev = head;
      while (prev != NULL && prev -> next != NULL)
      {
        cur = prev -> next;
        if (cur -> data < prev -> data)
        {
          next = cur -> data;
          cur -> data = prev -> data;
          prev -> data = next;
        }
        prev = prev -> next
    } }

void reverse( struct node **head)
  {
    struct node * prev = NULL;
    struct node * current = *head;
    while (current != NULL)
      {
        next = current -> Next;
        current -> next = prev;
    }
}
```

```
void concat ( struct node *head1 , struct node * head2) {
    struct node *prev = head2 ;
    while ( prev != NULL) {
      append ( &head1 , prev -> data) ;
      prev = prev -> next ;
    } }

int main ()
{
  struct node *head = NULL ;
  int choice
   bubble_sort (head) ;
   display (head) ;
   reverse (&head)
   display (head)
   append (&head2 , 76) ;
   concat (head , head2) ;
   display (head) ;
}
```

OUTPUT -

5 4 6 2 1 3

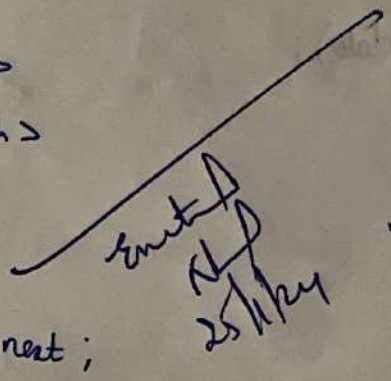
1 2 3 4 5 6

6 5 4 3 2 1

676 5 4 3 2 1

Program - 2

Stack implementation in singly linked list.



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

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

void append (struct node** head, int new_data) {
    struct node* new_node = (struct node*) malloc (sizeof (struct node));
    new_node -> data = new_data;
    new_node -> next = NULL;
    struct node* last = *head;
    if (*head == NULL)
        *head = new_node
    else {
        while (last -> next != NULL)
            last = last -> next;
        last -> next = new_node;
    } }

void display (struct node* head) {
    if (head == NULL) {
        printf ("Linked List empty");
        return;
    }
    printf ("Stack : ");
    while (head != NULL) {
        printf ("%d", head -> data);
    } }

void del_end (struct node *head) {
    if (head == NULL) {
        printf ("List empty");
        return;
    } struct node *last = head;
        prev = last;
        last = last -> next;
    }
```

```
int main() {
  struct node * head = NULL;
  int choice, a;
  while (choice < 4) {
     printf ("1. Push \n 2. Pop \n 3. Display ");
     scanf ("%d", &choice);
      switch (choice)
      {
       case 1: printf ("Enter value");
               scanf ("%d", &a);
               append (&head, a);
               display (head);
               break;
       case 2: del_end (head);
               display (head)
               break;
       default: break;
      }
   }
   return 0;
}
```

OUTPUT -

```
1. Push
2. Pop
3. Display
1
Enter the value
  23
Pushed .
```

Program - 3

Queue Functions Implementation in a linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
};
void append (struct node ** head, int new_data)
{
    struct node * new_node = (struct node*) malloc (sizeof (struct node));
    new_node -> data = new_data;
    new_node -> next = NULL;
    struct node * last = *head;
    if (*head == NULL)
        *head = new_node;
    else {
        while (last -> next != NULL)
            last = last -> next;
        last -> next = new_node;
    }}

void display (struct node* head) {
    if (head == NULL) {
        printf ("Linked list empty");
        return;
    } printf ("Queue: ");
    while (head != NULL) {
        printf ("%d", head -> data);

void del_head (struct node **head) {
        if (*head == NULL) {
            printf ("List is empty");
            return;
        }
```

```
    struct node * temp = (*head) -> next;
        free (*head);
        *head = temp;
}
int main() {
    struct node * head = NULL;
     int choice, a;
    while (choice <4) {
       printf (" 1. Push \n 2. Pop \n 3. Display ");
       scanf ("%d", &choice);
       switch (choice)
       {
        case 1: printf ("Enter values ");
                append (&head, a);
                display (head);
                 break;
         case 2: del_head (&head);
                 display (head);
                  break;
         case 3: display (head);
         default : break;
       } } }
```

OUTPUT -

1. Push
2. Pop
3. Display

Choice : 2

List is empty.

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

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

node * head = NULL;
int count = 0;

void insert (int data, int position);
void delete (int element);
void display ();
int main () {
    int data, choice, pos;
    printf ("1. Insert \n 2. Delete \n 3. Exit ");
    scanf ("%d", &choice");
    while (choice != 3) {
        if (choice == 1) {
        printf ("Enter data and position");
        scanf ("%d %d", &data, &pos);
        insert (data, pos);
        printf ("Count : %d \n", count);
        }
    display ();
    printf ("Enter choice : ");
    scanf ("%d", &choice);
    }
  return 0;
}
void insert (int data, int position) {
    if (position == 0) {
     node * new_node = malloc (sizeof (node));
```

Expted 11/2/24

```
new_node -> data = data;
new_node -> next = head;
new_node -> prev = NULL;
if (head != NULL) head -> prev = new_node;
 head = new_node;
 count ++;
 return;
}
else if ( position == count) {
    node* new_node = malloc ( sizeof (node));
    new_node => data = data;
    new_node -> next = NULL;
     node * temp = head;
    while ( temp -> next != NULL)
        temp = temp -> next;
     temp -> next = new_node;
     new_node => prev = temp;
     count ++;
     return;
    }
else if ( position > count || position < 0) {
     printf (" Unable to insert at given position \n");
     return;
 } else {
     node * temp = head;
     for (int i=0; i < position -1; i++)
         temp = temp -> next;
     node * new_node = malloc( sizeof (node));
     new_node -> data = data;
     count ++;
     return;
 } }

void delete (int element) {
    int position = 0; node * temp = head;
     if (head = NULL) {
         printf (" List is empty, cannot delete "); return;
```

```
if (position == 0) {
    node* temp = head
    head = temp;
    count--;
    return;
} else if (position == count-1) {
    node* temp = head;
    for(int i=1; i < count-1; i++)
        temp = temp -> next;
    count--;
    return;
} else if (position > count || position < 0) {
    printf ("Unable to delete at position ");
    return;
}}

void display () {
    node *temp = head;
    printf ("Linked List ");
    while (temp -> next != NULL) {
        printf ("%d", temp -> data);
}
```

Leetcode - Split Linked List

```
struct ListNode** splitListToParts(struct ListNode* head, int k,
    int* returnSize){
    struct ListNode* temp = head; int n=0;
    for(; temp != NULL; temp = temp->next, n++);
    struct ListNode** lists = (struct ListNode**) malloc(k* sizeof(
     struct ListNode*));
   for(int i=0; i<k; i++) lists[i]=NULL;
   int earlier_lists = n%k, size=n/k;
   int current =0; bool list_over = false;
   temp = head;
   *returnSize =k;
   for(int i = earlier_lists ;i>0; i--){
       struct ListNode* temp1 = temp;
       lists[current++] = temp;
       for(int j=0; j<size ; j++) temp1 = temp1->next;
       temp = temp1->next;
       temp1->next = NULL;
}
if(temp==NULL) return lists;
for(int i=0; i<k-earlier_lists; i++){
        struct ListNode* temp1=temp1=temp
        if(temp1==NULL) break;
        for(int j=0; j<size-1; j++) temp1 = temp1->next;
        lists[current++] = temp;
        temp = temp1->next;
        temp1->next=NULL;
   }
  return lists;
}
```

Week-8 15/2

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

typedef struct Node {
    int data;
    struct Node * left;
    struct Node * right
} node;

node * root = NULL;
void insert (node ** root, int data);
void preorder (node ** root);
void postorder (node ** root);
void inorder (node ** root);

int main () {
    int choice, data;
    insert (&root, 8);
    insert (&root, 3);
    insert (&root, 1);
    insert (&root, 6);
    insert (&root, 4);
    insert (&root, 7);
    insert (&root, 10);
    insert (&root, 14);
    insert (&root, 13);
    printf (" 1. Preorder \n 2. Inorder \n 3. Postorder \n 4. Exit \n Choice: ");
    scanf ("%d", &choice);
    while (choice != 4) {
        if (choice == 1) {
            preorder (&root);
            printf ("\n");
        }
```

NP
15/2/2024

```
else if (choice == 2) {
    inorder (&root);
    printf ("\n");
}
else if (choice == 3) {
    postorder (&root);
    printf ("\n");
}
printf ("Enter choice");
scanf ("%d", &choice);
}
}

void insert (node **root, int data) {
    if (*root == NULL) {
        node* new_node = malloc (sizeof (node));
        new_node -> data = data;
        new_node -> right = NULL;
        new_node -> left = NULL;
        *root = new_node;
        return;
    }
    if (data < (*root) -> data) {
        insert (&((*root) -> left), data);
    } else if (data > (*root) -> data) {
        insert (&((*root) -> right), data);
    }
    return;
}

void preorder (node **root) {
    if (*root != NULL) {
        printf ("%d", (*root) -> data);
        preorder (&((*root) -> left));
        preorder (&((*root) -> right));
    }
}
```

```
void postorder (node **root) {
    if (*root != NULL)
        postorder (&((*root)->left));
        postorder (&((*root)->right));
        printf ("%d", (*root)->data);
    }
}

void inorder (node **root) {
    if (*root != NULL) {
        inorder (&(*root)->left);
        printf ("%d", (*root)->data);
        inorder (&(*root)->right);
    }
}
```

OUTPUT

1. Preorder
2. Inorder
3. Postorder
4. Exit

Choice : 1

8 3 1 6 4 7 10 14 13

Choice : 2

1 3 4 6 7 8 10 13 14

Choice : 3

1 4 7 6 13 14 10 8

LeetCode-

## Rotate List

```
struct ListNode* rotateRight(struct ListNode* head, int k){
    struct ListNode *temp = head;
    if (head == NULL) return NULL;
    if (head->next == NULL) return head;
    if (k == 0) return head; int size = 1;
    temp->next = head;
    for(; temp->next != NULL; temp = temp->next, size++);
    k %= size;
    struct ListNode *temp1 = head;
    for (int i=0; i<(size-k-1); temp1 = temp1->next, i++);
    head = temp1->next;
    temp1->next = NULL;
    return head;
}
```

INPUT- head = [0,1,2], k = 4

Output = [2,0,1]

15/2/24.

Week - 9

BFS Traversal

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

#define size 7

void push(int a);
int pop();
void display();
void bfs(int graph[][7]);

int ffree = -1, rfree = -1;
int queue[size];

int main() {
    int adj_matrix[7][7] = {
        {0,1,0,1,0,0,0},
        {1,0,1,1,0,1,1},
        {0,1,0,1,1,1,0},
        {1,1,1,0,0,0,0},
        {0,0,1,0,0,0,1},
        {0,1,1,0,0,0,0},
        {0,1,0,0,1,0,0},
    };

    for (int i=0; i<7; i++) queue[i] = NULL;
    bfs(adj_matrix);
    return 0;
}

void bfs(int graph[][7]){
    int visited[7];
    for(int i=0; i<7; i++) visited[i] = 0;
```

```
push(0); visited[0]=1;
while(fhos != size){
    for(int i=0; i<7; i++){
        if(graph[queue[fhos]][i]==1 && visited[i]!=1){
            push(i);
            visited[i]=1;
        }
    }
    printf("%d", pop());
}
}

void push(int a){
    if(fhos==-1 && rhos==-1){
        queue[++rhos]=a;
        fhos++;
        return;
    }
    else if(rhos==size-1){
        printf("Queue overflow condition\n");
        return;
    }
    else{
        queue[++rhos]=a;
        return;
    }
}

int pop(){
    if(fhos==-1){
        printf("Queue Underflow condition\n");
    }
    int n=queue[fhos];
    queue[fhos]=(int)NULL;
    fhos++;
```

```
return n;
}
void display () {
    printf ("Queue ");
    for (int i=0 ; i< size ; i++)
        printf ("%d", queue[i]);
    printf ("\n");
}
```

OUTPUT-

0 1 3 2 5 6 4

DFS

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

#define size 7
int pos = -1;
int stack[size];
void push(int a);
int pop();
void display();
void dfs(int graph[][7]);
int main(){
    int adj_matrix[7][7] = {
        {0,1,0,1,0,0,0},
        {1,0,1,1,0,1,1},
        {0,1,0,1,1,1,0},
        {1,1,1,0,0,0,0},
        {0,0,1,0,0,0,1},
    };
    for(int i=0; i<7; i++) stack[i] = NULL;
    dfs(adj_matrix);
    return 0;
}

void dfs(int graph[][7]){
    int visited[7];
    for(int i=0; i<7; i++) visited[i]=0;
    push(0); visited(0)=1; printf("0");
    while(pos!=-1){
        bool new_node = false;
        for(int i=0; i<7; i++) visited[i]=0;
        push(0); visited(0)=1;
    }
```

```
void push (int a) {
    if (pos == size-1) {
        printf (" Stack Overflow Condition ");
        return ;
    }
    stack [++pos] = a;
}

int pop () {
    if ( pos == -1) {
        printf ( " Stack Underflow Condition ");
        return (int) NULL;
    }
    return stack [pos--] ;
}

void display () {
    for (int i = 0 ; i < size ; i++) {
        printf ("%d", stack [i]) ;
    }
    printf ("\n");
}
}
```

OUTPUT-

0 1 2 3 4 6 5

Sp.7
22/2/24

HackerRank

Swap Nodes

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

struct node {
    int data;
    struct node * left;
    struct node * right;
};

struct node * create_node (int val) {
    if (val == -1) {
        return NULL;
    }
    struct node * temp = (struct node*) malloc (sizeof (struct node));
    temp -> data = val;
    temp -> left = NULL;
    temp -> right = NULL;
    return temp;

void inorder (struct node * root) {
    if (!root) {
        return;
    }
    inorder (root -> left);
    printf ("%d", root -> data);
    inorder (root -> right);

int max (int a, b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
```

```
}
int height ( struct node * root) {
    if (!root){
        return 0;
    }
    return ( 1+ max( height(root -> left) , height (root -> right)));
}
void swap_nodes_at_level ( struct node * root, int lvl , int level,, int
height )
```

Sri
22/2/24

LAB-10

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10

int h[TABLE_SIZE] = {NULL};
void insert()
{
  int key, index, i, flag=0, hkey;
  printf("Enter a value to insert into hash table\n");
  scanf("%d", &key);
  hkey = key % TABLE_SIZE;
  for(i=0; i<TABLE_SIZE; i++)
    {
      index = (hkey + i) % TABLE_SIZE;
      if (h[index] == NULL)
        {
          h[index] = key
          break;
        }
    }
    if (i == TABLE_SIZE)
      printf("Element cannot be inserted\n");
}
void search()
{
  int key, index, i, flag=0, hkey;
  printf("Enter search element\n");
  scanf("%d", &key);
  hkey = key % TABLE_SIZE
  for(i=0; i<TABLE_SIZE; i++)
  {
    index = (hkey+i) % TABLE_SIZE;
    if (h[index] == key)
    {
```

```
        printf ("Value is found at index %d", index);
        break;
    }
}
if (i == TABLE_SIZE)
    printf ("The value is not found");
}
void display ()
{
 int i;
 printf (" Elements in the hash table are \n");
   for (i=0; i<TABLE_SIZE; i++)
    printf (" At index %d it value = %d", i, h[i]);
}
main()
{
 int opt, i;
 while (1)
 {
   printf (" Press 1. Insert it 2. Display it 3. Search it 4. Exit \n");
   scanf ("%d", &opt);
   switch (opt)
   {
     case 1: insert();
             break;
     case 2: display();
             break;
     case 3: search();
             break;
     case 4: exit(0);
   }
 }
}
```

OUTPUT -

Press 1. Insert
2. Display
3. Search
4. Exit

Enter value
23

Enter search element
3

Value is not found.

29/2/24