INDEX

NAME: Nikhilesh C STD.: ____ SEC.: D ROLL NO.: 181 SUB.: DS observation book.

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	07/12/2023	Week 0	1	10
2	21/12/2023	Week 1	6	
3.	28/12/2023	Week 2	14	10
4.	11/01/2024	Week 3	22	10
5.	18/01/2024	Week 4	32	10
6	11/01/2024	Leetcode - Stacks	30	5
7	25/01/2024	Week - 5	38	10
8	25/01/2024	Leetcode - Linked Lists	50	10
9	01/02/2024	Week - 6	53	10
10.	01/02/2024	Leetcode - Linked Lists	59	
11.	15/02/2024	Week - 7	60	10
12	22/02/2024	Week - 8	67	
13	22/02/2024	~~Leet~~ HackerRank - Trees	71	
14.	29/02/2024	Week - 9	75	N 29/2/24

07/12/2023

Week-0

Lab Program 1:

Output:

```
Enter 1 to create an account.
Enter 2 to withdraw an amount.
Enter 3 to deposit an amount.
Enter 4 for balance inquiry
Enter 0 to exit.

Enter an option:
1
Enter id number of the applicant
1234
Enter the initial amount to deposit:
2500
Enter a pin number.
1111
Account details recorded successfully.
Your account number is 3257.

Enter 1 to create an account.
Enter 2 to withdraw an amount.
Enter 3 to deposit an amount.
Enter 4 for balance enquiry.
Enter 0 to exit.

Enter an option:
2
Enter the account number:
3257
Enter the amount to withdraw:
500
```

Enter your pin:
1111
Please collect your cash.

Enter 1 to create an account.
Enter 2 to withdraw an amount.
Enter 3 to deposit an amount.
Enter 4 for balance inquiry.
Enter 0 to exit.

Enter an option:
3
Enter the account number:
3257
Enter the amount to deposit.
1000
Deposited successfully.

Enter 1 to create an account.
Enter 2 to withdraw an amount
Enter 3 to deposit an amount.
Enter 4 for balance inquiry.
Enter 0 to exit.

Enter an option
4
Enter the account number:
3257
The balance is 3000

Enter 1 to create an account.
Enter 2 to withdraw an amount.
Enter 3 to deposit an amount

Enter 4 for balance inquiry.
Enter 0 to exit.

Enter an option:
0

Exiting ...

2.
Enter the number of strings:
3
Enter the strings:
Basavanagudi
hebbal
bmsce
The lexicographically sorted strings are:
Basavanagudi
bmsce
hebbal.

3. Enter the number of elements in the array:
4
Enter the elements of the array:
a[0] = 45
a[1] = 56
a[2] = 69
a[3] = 82
Enter the element to check:
69
The element 69 is present in the array.

4. Enter the larger string:
BMSCE123456
Enter the substring:
E12
The substring is found in the larger string.

Enter the substring:
E21
The substring is not found in the larger string.

5. Enter the number of elements in the array:
6
a[0] = 1
a[1] = 2
a[2] = 1
a[3] = 3
a[4] = 2
a[5] = 1
Enter the number whose index of last occurrence has to be found:
1
The index of last occurrence of the number is 5.

6. Enter the number of elements in the array:
5
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
Enter the number to find:
4
The number is found at Position 4.

7. Enter the number of elements in the array:
5
Enter the elements in a sorted manner:
a[0] = 10
a[1] = 20
a[2] = 25
a[3] = 39
a[4] = 69
Enter the number to find:
39
The number is found at position 4.

8. Enter the number of elements in the array:
4
Enter the elements of the array:
a[0] = 45
a[1] = 69
a[2] = 96
a[3] = 2
The minimum element in the array is:
2
The ~~minimum~~ maximum element in the array is:
96.

Week - 1

1. Swapping two numbers using pointers.

```
#include <stdio.h>
void swapNumbers (int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main()
{
    int n1, n2;
    printf("Enter the first number: \n");
    scanf("%d", &n1);
    printf("Enter the second number: \n");
    scanf("%d", &n2);
    printf("Before Swapping: \n");
    printf("First number: %d \n", n1);
    printf("Second number: %d \n, n2);
    swapNumbers(&n1, &n2);
    printf("After Swapping: \n");
    printf("First number: %d \n", n1);
    printf("Second number: %d \n", n2);
    return 0;
}
```

Output:

```
Enter the first number : 2
Enter the second number : 3
Before Swapping :
First number : 2
```

Second number : 3

After Swapping :
First number : 3
Second number : 2

Date 16/12/23

2 Dynamic Memory allocation.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *arr1, *arr2;
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    arr1 = (int *) malloc(n* sizeof(int));
    if (arr1 == Null)
    {
        printf("Memory allocation failed for arr1\n");
        return 1;
    }
    printf("Memory allocation successful for arr1\n");
    printf("Enter the elements of arr1:\n");
    for (int i=0; i<n; i++)
    {
        printf("\n arr1[%d] = ", i);
        scanf("%d", &arr1[i]);
    }
    printf("Contents of arr1: ");
    for(int i=0; i<n; i++)
        printf("\n arr1[%d] = %d", i, arr1[i]);
    printf("\n");
```

```
// free (arr1);
// printf ("Memory deallocation successful for arr1\n");
printf ("Enter the size of the new array: \n");
scanf ("%d", &n);
arr2 = (int *) calloc (n, sizeof (int));
if (arr2 == Null)
{
    printf ("Memory allocation failed for arr2\n");
    return 1;
}
printf ("Memory allocation successful for arr2\n");
printf ("Contents of arr2 (initialized to zero): ");
for (int i=0; i<n ; i++)
    printf ("\n arr2[%d] = %d", i, arr2[i]);
printf ("\n");
printf ("Enter the size of the new array: \n");
scanf
printf ("Enter the new size for arr1: \n");
scanf ("%d", &n);
arr1 = (int *) realloc (arr1, n* sizeof (int));
if (arr1 == Null)
{
    printf ("Memory reallocation failed for arr1\n");
    return 1;
}
printf ("Memory reallocation successful for arr1\n");
printf ("Enter the elements of arr1: \n");
for (int i = 0; i<n; i++)
{
    printf ("\n arr1[%d] = ");
    scanf ("%d", &arr1[i]);
}
printf ("Contents of arr1: ");
```

```
    for(int i=0; i<n; i++)
        printf("\n arr1[%d] = %d", i, arr1[i]);
    printf("\n");
    free(arr1);
    free(arr2);
    printf("Memory deallocation successful for arr1 and
            arr2 \n");
    return 0;
}
```

Output:

```
Enter the size of the array: 3
Memory allocation successful for arr1
Enter the elements of arr1:

arr1[0] = 1
arr1[1] = 2
arr1[2] = 3
Contents of arr1:
arr1[0] = 1
arr1[1] = 2
arr1[2] = 3
Memory deallocation successful for arr1
Enter the size of the new array: 4
Contents of arr2 (initialized to zero):
arr2[0] = 0
arr2[1] = 0
arr2[2] = 0
arr2[3] = 0
Enter the new size for arr1:
4
Memory reallocation successful for arr1
```

Enter the elements of arr1:

arr1[1802406448] => 1
arr1[1802406408] => 2
arr1[1802406400] => 3

arr1[0] = 1
arr1[1] = 2
arr1[2] = 3
arr1[3] = 4

Memory deallocation successful for arr1 and arr2

3. Stack operations (Push, Pop & Display)

```
#include <stdio.h>
#include <stdlib.h>
#define Max_Size 5
struct stack
{
    int arr[Max_Size];
    int top;
};

void initialize(struct Stack *stack);
void push(struct Stack *stack);
void pop(struct Stack *stack);
void display(const struct Stack *stack);

int main()
{
    struct Stack stack;
    int c, cl = 0;
```

```
    initialize (& stack);
    while (c1 == 0)
    {
        printf ("Enter 1 to push an element into the stack\n");
        printf ("Enter 2 to pop an element from the stack\n");
        printf ("Enter 3 to display the contents of the stack\n");
        printf ("Enter 0 to exit\n");
        printf ("Enter your choice:");
        scanf ("%d", &c);
        switch (c)
        {
            case 0 : c1 = 1; break;
            case 1 : push (& stack); break;
            case 2 : pop (& stack); break;
            case 3 : display (& stack); break;
            default : printf ("Invalid choice\n"); break;
        }
    }
    return 0;
}
void initialize (struct Stack * stack)
{
    stack -> top = -1;
}
void push (struct Stack * stack)
{
    int value;
    printf ("Enter the element to push into the stack:\n");
    scanf ("%d", &value);
    if (stack -> top == Max_Size - 1)
    {
        printf ("Stack Overflow : Cannot push %d, stack is
                full.\n", value);
    }
```

```
    else
    {
        stack -> arr[++(stack -> top)] = value;
        printf ("Pushed %d onto the stack.\n", value);
    }
}
void pop (struct Stack *stack)
{
    int poppedElement;
    if (stack -> top == -1)
    {
        printf ("Stack Underflow: Cannot pop from an empty
                 stack.\n");
    }
    else
    {
        poppedElement = stack -> arr[(stack -> top)--];
        printf ("The popped element is %d \n", poppedElement);
    }
}
void display (const struct Stack *stack)
{
    if (stack -> top == -1)
        printf ("Stack is empty.\n");
    else
    {
        printf ("Stack contents: ");
        for (int i = 0; i <= stack -> top; i++)
            printf ("%d", stack -> arr[i]);
        printf ("\n");
    }
}
```

Output :

```
Enter 1 to push an element into the stack
Enter 2 to pop an element from the stack
Enter 3 to display the contents of the stack
Enter 0 to Exit.
#
Enter your choice:1
Enter the element to push into the stack:
2
Pushed 2 into the stack.

Enter your choice : 2
The popped element is : 2

Enter your choice : 3
Stack is empty

Enter your choice : 0
Exiting ...
```

26/12/23

28/12/2023

Week - 2

1. Converting an infix expression to a postfix expression

```
#include <stdio.h>
#include <stdlib.h>
#define Max 100
char stack[Max];
int top = -1;
void push(char item)
{
    if (top == Max-1)
    {
        printf("Stack Overflow\n");
        exit(EXIT_FAILURE);
    }
    stack[++top] = item;
}
char pop()
{
    if (top == -1)
    {
        printf("Stack Underflow\n");
        exit(EXIT_FAILURE);
    }
    return stack[top--];
}
int precedence(char operator)
{
    switch(operator)
    {
        case '^': return 3;
        case '*':
```

```
        case '/':
            return 2;
        case '+':
        case '-':
            return 1;
        default: return 0;
    }
}
int main()
{
    char infix[Max];
    printf("Enter a valid parenthesized infix expression: ");
    scanf("%s", infix);
    char postfix[Max];
    int i, j;
    i = j = 0;
    while (infix[i] != '\0')
    {
        char symbol = infix[i];
        if ((symbol >= 'a' && symbol <= 'z') ||
            (symbol >= 'A' && symbol <= 'z'))
            postfix[j++] = symbol;
        else if (symbol == '(')
            push(symbol);
        else if (symbol == ')')
        {
            while (top != -1 && stack[top] != '(')
                postfix[j++] = pop();
            if (top == -1)
            {
                printf("Invalid expression due to paranthesis
                    mismatch\n");
                exit(EXIT_FAILURE);
            }
```

```
            pop();
        }
        else
        {
            while (top != -1 && precedence(stack[top]) >=
                                precedence(symbol))
                postfix[j++] = pop();
            push(symbol);
        }
        i++;
    }
    while (top != -1)
    {
        if (stack[top] == '(')
        {
            printf("Invalid Expression due to parenthesis
                    mismatch\n");
            exit(EXIT_FAILURE);
        }
        postfix[j++] = pop();
    }
    postfix[j] = '\0';
    printf("Postfix Expression : %s\n", postfix);
    return 0;
}
```

Output:

Enter a valid parenthesized infix expression:
A*B+C*D-E

Postfix Expression : AB*CD*+E-

2. Evaluation of a postfix expression.

```
#include <stdio.h>
#include <stdlib.h>
#define Max 100
int stack[Max];
int top = -1;
void push (int element)
{
    if (top == Max-1)
    {
        printf ("Stack Overflow\n");
        exit (EXIT_FAILURE);
    }
    stack[++top] = element;
}
int pop()
{
    if (top == -1)
    {
        printf ("Stack Underflow\n");
        exit (EXIT_FAILURE);
    }
    return stack[top--];
}
int main()
{
    char postfix[100];
    printf ("Enter a postfix expression: ");
    scanf ("%s", postfix);
    for (int i=0; postfix[i]!='\0'; i++)
    {
        if (isdigit (postfix[i]))
```

```
            push(postfix[i] - '0');
        else
        {
            int op2 = pop();
            int op1 = pop();
            switch(postfix[i])
            {
                case '+' : push(op1 + op2); break;
                case '-' : push(op1 - op2); break;
                case '*' : push(op1 * op2); break;
                case '/' : push(op1 / op2); break;
                default : printf("Invalid operator\n");
                          exit(EXIT_FAILURE);
            }
        }
    }
    int result = pop();
    printf("Result: %d\n", result);
    return 0;
}
```

Output:

```
Enter a postfix expression : 12*34*+5-
Result : 9
```

3 Linear Queue Implementation.

```
#include <stdio.h>
#define SIZE 5
int front = -1, rear = -1;
int queue[SIZE];
```

```
int main()
{
    int c, elem, cl=0;
    while (cl==0)
    {
        printf("\nEnter 1 to insert into the queue\n");
        printf("Enter 2 to delete from the queue\n");
        printf("Enter 3 to display the contents of the
            queue\n");
        printf("Enter 0 to exit\n");
        printf("\n Enter your choice:\n");
        scanf("%d", &c);
        switch(c)
        {
            case 0: printf("Exiting...\n"); cl=1; break;
            case 1:
                printf("Enter the element to be inserted:\n");
                scanf("%d", &elem);
                enqueue(elem);
                break;
            case 2: dequeue(); break;
            case 3: display_q(); break;
            default: printf("Invalid choice\n"); break;
        }
    }
    return 0;
}

void enqueue(int elem)
{
    if(rear == SIZE-1)
    {
        printf("Queue is full. Cannot insert\n");
        exit(0);
    }
```

~~Singly Linked list~~

```
    queue[++rear] = elem;
    printf("Element %d inserted successfully\n", elem);
}

void dequeue()
{
    if (front == -1 || front > rear)
    {
        printf("Queue is empty. Cannot delete\n");
        exit(0);
    }
    printf("The element deleted is %d\n", queue[front]);
    front++;
}

void display_q()
{
    if (front == -1)
    {
        printf("Queue is empty\n");
        exit(0);
    }
    printf("The elements of the queue are:\n");
    for(int i = front; i <= rear; i++)
        printf("%d", queue[i]);
}
```

Output:

```
Enter 1 to insert into the queue
Enter 2 to delete from the queue
```

Enter 3 to display the contents of the queue.
Enter 0 to exit

Enter your choice:
1
Enter the element to be inserted:
1
Element 1 inserted successfully

Enter your choice:
1
Enter the element to be inserted:
2
Element 2 inserted successfully

Enter your choice:
1
Enter the element to be inserted:
3
Element 3 inserted successfully

Enter your choice:
3
The elements of the queue are:
123

Enter your choice:
2
The element deleted is 1

Enter your choice:
3
The elements of the queue are:
23

11/01/2024

Week - 3

1. Circular Queue Implementation

```
# include < stdio.h >
# define SIZE 5
int front = -1, rear = -1;
int CQ[SIZE];
int main ()
{
    int c, cl = 0;
    while (cl == 0)
    {
        printf ("\n Enter 1 to insert into the circular queue \n");
        printf ("Enter 2 to delete from the circular queue \n");
        printf ("Enter 3 to display the contents of the
                circular queue \n");
        printf (" Enter 0 to exit \n");
        printf ("\n Enter your choice : \n");
        scanf ("%d", &c);
        switch (c)
        {
            case 0: printf (" Exiting ... \n"); cl = 1; break;
            case 1 : enqueue (); break;
            case 2 : dequeue (); break;
            case 3 : display_q (); break;
            default : printf ("Invalid choice \n");
        }
    }
    return 0;
}

void enqueue ()
{
```

```
    int elem;
    if (rear == SIZE - 1)
    {
        printf ("Queue is full. Cannot insert\n");
        exit (0);
    }
    if (front == -1)
    {
        front = 0;
        rear = 0;
    }
    printf ("Enter the element to be inserted : \n");
    scanf ("%d", &elem);
    ca[rear] = elem;
    printf ("Element %d inserted successfully\n", elem);
    rear = (rear + 1) % SIZE;
}

void dequeue ()
{
    if (front == -1)
    {
        printf ("Queue is empty \n");
        exit (0);
    }
    printf ("The elements of the queue are : \n");

    printf ("The element deleted is %d \n", ca[front]);
    front = (front + 1) % SIZE;
}

void display_q ()
{
```

```
    if (front == -1)
    {
        printf ("Queue is empty \n");
        exit (0);
    }
    printf ("The elements of the queue are : \n");
    for (int i = front ; i <= rear ; i++)
        printf ("%d", CQ[i]);
}
```

Output :

Enter 1 to insert into the circular queue
Enter 2 to delete from the circular queue
Enter 3 to display the contents of the circular queue
Enter 0 to exit.

Enter your choice :
1
Enter the element to be inserted :
1
Element 1 inserted successfully

Enter your choice :
1
Enter the element to be inserted :
2
Element 2 inserted successfully

Enter your choice :
1
Enter the element to be inserted :

3
Element 3 inserted successfully.

Enter your choice:
3
The elements of the queue are:
123

Enter your choice:
2
The element deleted is 1

Enter your choice:
3
The elements of the queue are
23

2. Singly linked list - Insert and display implementation.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *next;
};
int main()
{
    struct Node *head = NULL;
    int c = 0, a, b;
    while(c!=3)
    {
        printf("\n Enter 1 to insert \n");
```

```
printf ("Enter 2 to display\n");
printf ("Enter 3 to exit\n");
printf ("\nEnter your choice :\n");
scanf ("%d", &c);
switch (c)
{
    case 1:
        printf ("Enter the value to be inserted :\n");
        scanf ("%d", &a);
        int c2;
        printf ("\nEnter 1 to insert at the front\n");
        printf ("Enter 2 to insert after a given node\n");
        printf ("Enter 3 to insert at the end\n");
        scanf ("%d", &c2);
        switch (c2)
        {
            case 1:
                insertFront (&head, a);
                break;
            case 3:
                insertEnd (&head, a);
                break;
            case 2:
                printf ("Enter the value after which to
                        insert:\n");
                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");
                else
                {
```

```
                    insertMiddle (temp, a);
                }
                break;
            default : printf ("Invalid choice \n"); break;
        }
    case 2 :
        display (head) ; break ;
    default : printf (" Invalid choice \n") ; break;
    }
}
    return 0;
}

void insertFront ( struct Node ** head, int new_data )
{
    struct Node * new_node = (struct Node *) malloc ( sizeof
                                (struct Node));
    new_node -> data = new_data;
    new_node -> next = (*head);
    (*head) = new_node ;
}

void insertMiddle ( struct Node *previous, int new_data)
{
    if (previous == NULL)
    {
        printf ("The previous node entered cannot be NULL\n");
        return ;
    }
    struct Node * new_node = (struct Node *) malloc ( sizeof
                                (struct Node));
    new_node -> data = new_data ;
    new_nod -> next = previous -> next ;
```

```
    previous -> next = new_node;
}

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

void display (struct Node *node)
{
    printf("The contents of the list are: \n");
    while (node != NULL)
    {
        printf("%d", node -> data);
        node = node -> next;
    }
    printf("\n");
}
```

Output :

```
Enter 1 to insert
Enter 2 to display
Enter 3 to exit

Enter your choice:
1
Enter the value to be inserted:
1
Enter 1 to insert at the front
Enter 2 to insert after a given node
Enter 3 to insert at the end
Enter your choice:
1
The contents of the list are:
1

Enter the value to be inserted:
2
:
Enter your choice:
1
The contents of the list are:
21

Enter the value to be inserted:
3
:
Enter your choice:
2
Enter the value after which to insert:
2
The contents of the list are:
231
```

Enter the value to be inserted:
4
:
Enter your choice:
3
The contents of the list are:
2314

Leetcode Problem:

```
typedef struct {
    int array[20000];
    int min[20000];
    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;
    }
```

```
        else {
            obj -> min[++obj -> top2] = m
        }
        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 -> min[obj -> top2];
}

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

Sneha
18/1/24

Output: All test cases passed

18/01/2024

Week - 4

1. Singly Linked list - Delete & Display implementation

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

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

int main()
{
    struct Node* head = NULL;
    insertEnd(&head, 1);
    insertEnd(&head, 2);
    insertEnd(&head, 3);
    insertEnd(&head, 4);
    insertEnd(&head, 5);
    insertEnd(&head, 6);
    printf("Initial List: \n");
    display_q(head);
    int c, cl = 0;
    while(cl == 0)
    {
        printf("Enter 1 to delete from the beginning \n");
        printf("Enter 2 to delete at the end \n");
        printf("Enter 3 to delete from a specific
                position \n");
        printf("Enter 4 to display \n");
        printf("Enter 0 to exit \n");
        printf("\n Enter your choice: \n");
```

```
        scanf("%d", &c);
        switch(c)
        {
            case 0: printf("Exiting...\n"); cl=1; break;
            case 1: delete_beg(&head); display_q(head); break;
            case 2: delete_end(&head); display_q(head); break;
            case 3: delete_mid(&head); display_q(head); break;
            case 4: display_q(head); break;
            default: printf("Invalid choice. Enter again!\n");
                     break;
        }
    }
    return 0;
}

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

void delete_end(struct Node** head)
{
    struct Node* ptr;
    struct Node* ptr1 = NULL;
```

```
    if (*head == NULL)
        printf ("List is empty \n");
    else if ((*head) -> next == NULL)
    {
        free (*head);
        *head = NULL;
        printf (" Deleted the only node in the list \n");
    }
    else
    {
        ptr = *head;
        while (ptr -> next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr -> next;
        }
        ptr1 -> next = NULL;
        free (ptr);
        printf (" Deleted the last node from the list \n");
    }
}

void delete_mid (struct Node **head)
{
    struct Node* ptr;
    struct Node* ptr1 = NULL;
    int loc;
    printf ("Enter the location of the node to be
            deleted : \n");
    scanf ("%d", &loc);
    ptr = *head;
    for (int i=0; i< loc; i++)
    {
```

```
        ptr1 = ptr;
        ptr = ptr -> next;
        if (ptr == NULL)
        {
            printf ("Less elements than required in the list\n");
            return;
        }
    }
    ptr1 -> next = ptr -> next;
    free (ptr);
    printf ("Node deleted from position %d \n", loc);
}

void display_q (struct Node* head)
{
    struct Node* current = head;
    if (current == NULL)
    {
        printf ("The list is empty \n");
        return;
    }
    else
    {
        printf ("The contents of the list are: \n");
        while (current != NULL)
        {
            printf ("%d", current -> data);
            current = current -> next;
        }
        printf ("\n");
    }
}
```

```
struct Node* createLL(int data)
{
    struct Node* newNode = (struct Node*) malloc(sizeof(
                                        struct Node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}

void insertEnd(struct Node** head, int data)
{
    struct Node* newNode = createLL(data);
    if (*head == NULL)
        *head = newNode;
    else
    {
        struct Node* current = *head;
        while (current -> next != NULL)
            current = current -> next;
        current -> next = newNode;
    }
}
```

Output

```
Initial list:
The contents of the list are:
1 2 3 4 5 6

Enter 1 to delete from the beginning
Enter 2 to delete at the end
Enter 3 to delete from a specific position.
Enter 4 to display
Enter 0 to exit
```

Enter your choice :
1
Node deleted from beginning
The contents of the list are :
23456

Enter your choice :
2
Deleted the last node from the list
The contents of the list are :
2345

Enter your choice :
3
Enter the location of the node to be deleted :
2
Node deleted from position 2
The contents of the list are :
235

18/1/24

25/10/2022

Week - 5

1. * Sort, Reverse and concatenation of singly linked list

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

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

void sort_list (struct Node *head)
{
    struct Node *fr, *q;
    int temp;
    for (fr = head; fr != NULL; fr = fr -> Next)
    {
        for (q = fr -> next; q != NULL; q = q -> ~~Next~~ next)
        {
            if (fr -> data > q -> data)
            {
                temp = q -> data;
                q -> data = fr -> data;
                fr -> data = temp;
            }
        }
    }
}

void reverse_list (struct Node **head)
{
    struct Node *cur = *head;
```

```
    struct Node *pre = NULL;
    struct Node *next = NULL;
    while (cur != NULL)
    {
        next = cur -> next;
        cur -> next = pre;
        pre = cur;
        cur = next;
    }
    *head = pre
}
void concat_list (struct Node *head1, struct Node *head2)
{
    struct Node *head3;
    struct Node *temp;
    if (head1 == NULL)
    {
        head3 = head2;
        display_list (head3);
    }
    else if (head2 == NULL)
    {
        head3 = head1;
        display_list (head3);
    }
    else
    {
        temp = head1;
        head3 = head1;
        while (temp -> next != NULL)
            temp = temp -> next;
        temp -> next = head2;
        display_list (head3);
    }
}
```

```
void display_list (struct Node *head)
{
    struct Node *current = head;
    if (current == NULL)
    {
        printf ("The list is empty \n");
        return;
    }
    else
    {
        printf ("The contents of the list are: \n");
        while (current != NULL)
        {
            printf ("%d", current -> data);
            current = current -> next;
        }
        printf ("\n");
    }
}

struct Node* createLinkedList (int data)
{
    struct Node *newNode = (struct Node*) malloc
                           (sizeof (struct Node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}

void insertEnd (struct Node **head, int data)
{
    struct Node **newNode = createLinkedList (data);
    if (*head == NULL)
```

```
    {
        *head = newNode;
    }
    else
    {
        struct Node *current = *head;
        while (current -> next != NULL)
        {
            current = current -> next;
        }
        current -> next = newNode
    }
}

int main()
{
    struct Node *head = NULL;
    insertEnd (&head, 3);
    insertEnd (&head, 1);
    insertEnd (&head, 4);
    insertEnd (&head, 6);
    insertEnd (&head, 5);
    struct Node *head1 = NULL;
    insertEnd (&head1, 1);
    insertEnd (&head1, 3);
    insertEnd (&head1, 4);
    insertEnd (&head1, 2);
    struct Node *head2 = NULL;
    insertEnd (&head2, 3);
    insertEnd (&head2, 1);
    insertEnd (&head2, 4);
    insertEnd (&head2, 2);
    struct Node *head3 = NULL;
```

```
printf ("Initial list :\n");
display_list (head);
int c, cl = 0;
while (cl == 0)
{
    printf ("\nEnter 1 to sort the linked list\n");
    printf ("Enter 2 to reverse the linked list\n");
    printf ("Enter 3 to concatenate 2 linked lists\n");
    printf ("Enter 4 to display\n");
    printf ("Enter 0 to exit\n");
    printf ("\nEnter your choice :\n");
    scanf ("%d", &c);
    switch (c)
    {
        case 0: printf ("Exiting...\n"); cl=1; break;
        case 1: sort_list (head); display_list (head); break;
        case 2: reverse_list (&head);
                display_list (head);
                break;
        case 3:
                printf ("The 2 lists being concatenated are :\n");
                display_list (head1);
                printf ("and\n");
                display_list (head2);
                printf ("Concatenated list :\n");
                concat_list (head1, head2);
                break;
        case 4: display_list (head); break;
        default: printf ("Invalid choice\n"); break;
    }
}
return 0;
}
```

Output :

Initial List :
The contents of the list are :
314 265

Enter 1 to sort the linked list
Enter 2 to reverse the linked list
Enter 3 to concatenate 2 linked lists
Enter 4 to display
Enter 0 to exit

(i) Enter your choice :
1
The contents of the list are :
123456

(ii) Enter your choice :
2
The contents of the list are :
654321

(iii) Enter your choice :
3
The 2 lists being concatenated are :
The contents of the list are :
1342
and
The contents of the list are :
3142
Concatenated list :
The contents of the list are :
13423142

2. Stack and Queue Operations using singly linked lists.

Ans:

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

void push_list (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 pop_listStack (struct Node *head)
{
    if (head == NULL)
    {
        printf ("List is empty \n");
        return;
    }
```

→ Add the case for 1 element in the list

```
    struct Node *last = *head;
    struct Node *pre;
    while (last -> next != NULL)
    {
        pre = last;
        last = last -> next;
    }
    free (last);
    pre -> next = NULL;
}

void enqueue_list (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 dequeue_list (struct Node **head)
{
    if (*head == NULL)
    {
        printf ("List is empty \n");
        return;
    }
```

→ Add the case of one element in the list.

```
    struct Node *temp = (*head) -> next;
    free (*head);
    *head = temp;
}

void display_ListStack (struct Node *head)
{
    if (head == NULL)
    {
        printf ("List is empty \n");
        return;
    }
    printf (" Stack: \n");
    while (head != NULL)
    {
        printf ("%d", head -> data);
        head = head -> next;
    }
    printf ("\n");
}

void display_listQueue (struct Node *head)
{
    if (head == NULL)
    {
        printf ("List is empty \n");
        return;
    }
    printf ("Queue : \n");
    while (head != NULL)
    {
        printf ("%d", head -> data);
        head = head -> next;
    }
```

```
    printf("\n");
}

int main()
{
    struct Node *head = NULL;
    struct Node *head1 = NULL;
    int c, c1 = 0, elem1, elem2;
    while (c1 == 0)
    {
        printf("\nEnter 1 to push \n");
        printf("Enter 2 to pop\n");
        printf("Enter 3 to display stack list\n");
        printf("Enter 4 to enqueue \n");
        printf("Enter 5 to dequeue \n");
        printf("Enter 6 to display queue list\n");
        printf("Enter 0 to exit\n");
        printf("\n Enter your choice\n");
        scanf("%d", &c);
        switch (c)
        {
            case 0: printf("Exiting ...\n"); break;
            case 1:
                printf("Enter the value to push:\n");
                scanf("%d", &elem1);
                push_list(&head, elem1);
                display_ListStack(head);
                break;
            case 2:
                pop_listStack(head);
                display_ListStack(head);
                break;
            case 3: display_ListStack(head); break;
```

```
            case 4:
                printf("Enter the value to enqueue:\n");
                scanf("%d", &elem2);
                enqueue_list(&head1, elem2);
                display_listQueue(head1);
                break;
            case 5:
                dequeue_list(&head1);
                display_listQueue(head1);
                break;
            case 6 : display_listQueue(head1); break;
            default : printf("Invalid choice\n"); break;
        }
    }
    return 0;
}
```

Output :

```
Enter 1 to push
Enter 2 to pop
Enter 3 to display stack list
Enter 4 to enqueue
Enter 5 to dequeue
Enter 6 to display Queue list
Enter 0 to exit

(i:) Enter your choice:
1
Enter the value to push:
1
Stack:
1
```

(i) Enter your choice :
1
Enter the value to push :
2
Stack :
12

(ii) Enter your choice :
2
Stack :
1

(iii) Enter your choice :
3
stack :
1

(iv) Enter your choice :
4
Enter the value to enqueue :
1
Queue :
1

Enter your choice :
4
Enter the value to enqueue :
2
Queue :
12

(v) Enter your choice :
5

Queue :
2

(vi) Enter your choice :
6
Queue :
6.

Leetcode [Singly linked list] :

```
void append (struct ListNode **head, int val)
{
    struct ListNode *new_node = (struct ListNode *) malloc
                                (sizeof (struct ListNode));
    new_node -> val = val;
    new_node -> next = NULL;
    struct ListNode *prev = *head;
    if (prev == NULL)
    {
        *head = new_node;
        return;
    }
    while (prev -> next != NULL)
        prev = prev -> next;
    prev -> next = new_node;
}

void mid (struct ListNode *prev, int val)
{
    struct ListNode *new_node = (struct Listnode *) malloc
                                (sizeof (struct ListNode));
    new_node -> val = val;
    new_node -> next = prev -> next;
```

```
    prev -> next = new_node;
}

void push (struct ListNode **head, int value)
{
    struct ListNode *new_node = (struct ListNode *) malloc
                                (sizeof(struct ListNode));
    new_node -> val = value;
    new_node -> next = (*head);
    *head = new_node;
}

struct ListNode* reverseBetween (struct ListNode *head, int left,
                                 int right)
{
    struct ListNode *newhead = NULL;
    int i = 1;
    struct ListNode *cur = head;
    struct ListNode *prev = newhead;
    while (cur != NULL)
    {
        if (left <= i && right >= i)
        {
            if (prev == NULL)
            {
                append (&newhead, cur -> val);
                prev = newhead;
                cur = cur -> next;
                i++;
                continue;
            }
            else if (left == 1)
                push (&newhead, cur -> val);
```

```
            else
            {
                append(&newhead, cur->val);
                prev = (prev == NULL) ? newhead : prev->next;
            }
            i++;
            cur = cur->next;
        }
        return newhead;
}

void display(struct ListNode *head)
{
    if (head == NULL)
    {
        printf("Linked list is empty.\n");
        return;
    }
    printf("Linked list : ");
    while (head != Null)
    {
        printf("%d", head->val);
        head = head->next;
    }
    printf("\n");
}
```

Output :

All test cases passed.

30/1/24

01/02/2024

Week - 6

1. Doubly Linked List - Creation, insertion to the left of a node, deletion of a node and displaying the contents.

Pms.

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

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

struct Node *createNode(int value)
{
    struct Node * newNode = (struct Node *) malloc (sizeof
                            (struct Node));
    if (newNode == NULL)
    {
        printf (" Memory allocation failed \n");
        exit (1);
    }
    newNode -> data = value;
    newNode -> prev = NULL;
    newNode -> next = NULL;
    return newNode;
}

void insertLeft (struct Node **head, int value, int targetValue)
{
    struct Node *newNode = createNode (value);
```

```
struct Node *current = *head;
while (current != NULL && current -> data != targetValue)
{
    current = current -> next;
}
if (current == NULL)
{
    printf ("Node with value %d not found \n", targetValue);
    free(newNode);
    return;
}
if (current -> prev != NULL)
{
    current -> prev -> next = newNode;
    newNode -> prev = current -> prev;
}
else
    *head = newNode;
newNode -> next = current;
current -> prev = newNode;
}

void deleteNode (struct Node **head, int value)
{
    struct Node * current = *head;
    while (current != NULL && current -> data != value)
        current = current -> next;
    if (current == NULL)
    {
        printf ("Node with value %d not found \n", value);
        return;
    }
    if (current -> prev != NULL)
```

```
        current->prev->next = current->next;
    else
        *head = current->next;
    if (current->next != NULL)
        current->next->prev = current->prev;
    free(current);
}

void displayList(struct Node *head)
{
    struct Node *current = head;
    while (current != NULL)
    {
        printf("%d", current->data);
        current = current->next;
    }
    printf("\n");
}

int main()
{
    struct Node *head = NULL;
    int choice, value, targetValue;
    do
    {
        printf("Enter 1 to create a doubly linked list\n");
        printf("Enter 2 to insert a node to the left of a node\n");
        printf("Enter 3 to delete a node based on a
                specific value\n");
        printf("Enter 4 to display the contents of the list\n");
        printf("Enter 5 to exit\n");
        printf("\nEnter your choice:\n");
        scanf("%d", &choice);
```

```
switch (choice)
{
    case 1:
        if (head != NULL)
            printf ("Doubly linked list already created \n");
        else
        {
            printf ("Enter the value for the node: \n");
            scanf ("%d", &value);
            head = createNode (value);
            printf ("Doubly linked list created
                    successfully \n");
        }
        break;

    case 2:
        if (head == NULL)
            printf ("List is empty. Please create a list
                    first \n");
        else
        {
            printf ("Enter the value to insert: \n");
            scanf ("%d", &value);
            printf ("Enter the value of the node to the
                    left of which to insert: \n");
            scanf ("%d", &targetValue);
            insertLeft (&head, value, targetValue);
        }
        break;
    case 3:
        if (head == NULL)
            printf ("List is empty. Please create a list
                    first. \n");
```

```
            else
            {
                printf("Enter the value to delete:\n");
                scanf("%d", &value);
                deleteNode(&head, value);
            }
            break;
        case 4:
            if (head == NULL)
                printf("List is empty\n");
            else
            {
                printf("The contents of the list are:\n");
                displayList(head);
            }
            break;
        case 5:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice\n");
            break;
    }
} while (choice != 5);
return 0;
}
```

Output :

```
Enter 1 to create a doubly linked list
Enter 2 to insert a node to the left of a node
Enter 3 to delete a node based on a specific value
Enter 4 to display the contents of the list.
```

Enter 5 to exit

(i.) Enter your choice :
1
Enter the value for the node :
3
Doubly linked list created successfully.

(ii.) Enter your choice :
2
Enter the value to insert :
2
Enter the value of the node to the left of which to insert :
3

Enter your choice :
2
Enter the value to insert :
1
Enter the value of the node to the left of which to insert :
2

(iii.) Enter your choice :
4
The contents of the list are :
123

(iv.) Enter your choice :
3
Enter the value to delete :
2

(v.) Enter your choice :

4
The contents of the list are:
13

Sneha
1/2/24

Leetcode - Singly linked list :

```
struct ListNode ** splitListToParts(struct ListNode* head,
                    int k, int* returnSize) {

    struct ListNode *current = head;
    int length = 0;
    while (current != NULL) {
        length++;
        current = current->next;
    }
    int part_size = length / k;
    struct ListNode **result = (struct ListNode **) malloc
            (k * sizeof(struct ListNode *));
    current = head;
    for (int i = 0; i < k; i++) {
        struct ListNode *part_head = current;
        int part_length = part_size + (i < extra_nodes ? 1 : 0);
        for (int j = 0; j < part_length - 1 && current; j++) {
            current->next = NULL;
            result[i] = part_head;
            current = next_node;
            current = current->next;
        }
        if (current) {
            struct ListNode **next_node = current->next;
```

int extra_nodes = length % k;

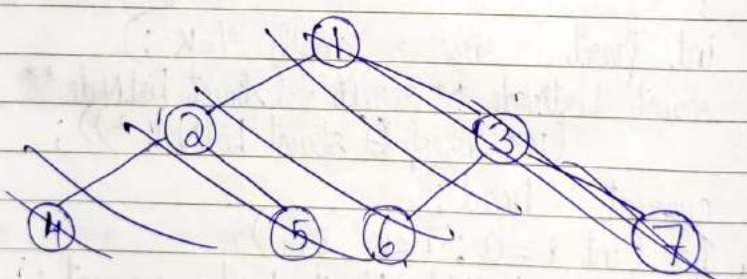
```
            current -> next = NULL;
            result[i] = part_head;
            current = next_node;
        }
        else {
            result[i] = NULL;
        }
    }
    returnSize = k;
    return result;
}
```

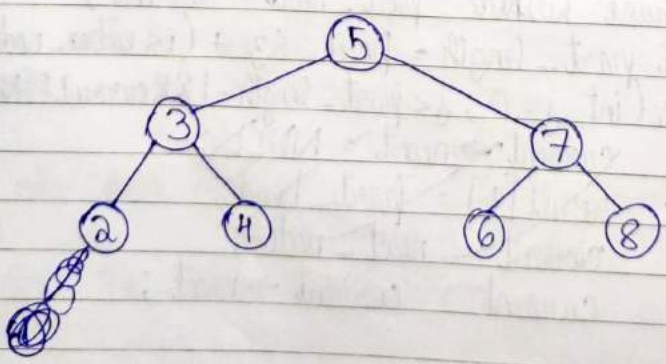
Output:

All testcases passed.

15-02-2024

Week - 7





1. Binary Search Tree:

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

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

struct Node* createNode(int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(
                                struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int value)
{
    if(root == NULL)
        return createNode(value);
    if(value < root->data)
        root->left = insert(root->left, value);
    else if(value > root->data)
        root->right = insert(root->right, value);
    return root;
}
```

```
void inOrderTraversal (struct Node * root)
{
    if (root != NULL)
    {
        inOrderTraversal (root -> left);
        printf ("%d", root -> data);
        inOrderTraversal (root -> right);
    }
}

void preOrderTraversal (struct Node * root)
{
    if (root != NULL)
    {
        printf ("%d ", root -> data);
        preOrderTraversal (root -> left);
        preOrderTraversal (root -> right);
    }
}

void postOrderTraversal (struct Node* root)
{
    if (root != NULL)
    {
        postOrderTraversal (root -> left);
        postOrderTraversal (root -> right);
        printf ("%d ", root -> data);
    }
}

void display (struct Node* root)
{
    printf ("Elements in the tree : \n");
```

```
    inOrderTraversal(root);
    printf("\n");
}

int main()
{
    struct Node* root = NULL;
    int c, value;
    do {
        printf("\nEnter 1 to insert an element\n");
        printf("Enter 2 to display elements\n");
        printf("Enter 3 to perform In-order traversal\n");
        printf("Enter 4 to perform Pre-order traversal\n");
        printf("Enter 5 to perform Post-order traversal\n");
        printf("Enter 0 to exit\n");
        scanf("%d", &c);
        switch(c)
        {
            case 1:
                printf("Enter the element to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                display(root);
                break;
            case 3:
                printf("In-order traversal\n");
                inOrderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Pre-order Traversal\n");
```

```
            preOrderTraversal(root);
            printf("\n");
            break;
        case 5:
            printf("Post-Order traversal: ");
            postOrderTraversal(root);
            printf("\n");
            break;
        default:
            printf("Invalid choice \n");
            break;
    }
} while(c!=0);
return 0;
}
```

Output:

```
Enter 1 to insert an element
Enter 2 to Display elements
Enter 3 to perform In-order traversal
Enter 4 to perform Pre-order traversal
Enter 5 to perform Post-order traversal
Enter 0 to exit

(i) Enter your choice : 1
Enter the element to insert : 5

Enter your choice : 1
Enter the element to insert : 3

Enter your choice : 1
Enter the element to insert : 7
```

Enter your choice : 1
Enter the element to insert : 7

Enter your choice : 1
Enter the element to insert : 2

Enter your choice : 1
Enter the element to insert : 4

Enter your choice : 1
Enter the element to insert : 6

Enter your choice : 1
Enter the element to insert : 8

(ii.) Enter your choice : 2
Elements in the tree : 2 3 4 5 6 7 8

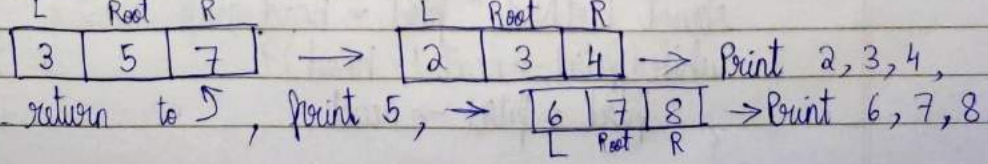
(iii.) Enter your choice : 3
In-Order traversal : 2 3 4 5 6 7 8

(iv.) Enter your choice : 4
Pre-Order traversal : 5 3 2 4 7 6 8

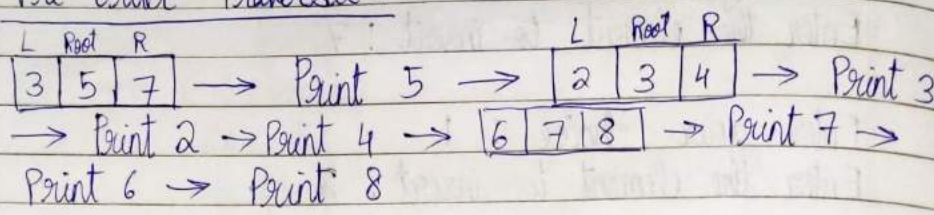
(v.) Enter your choice : 5
Post-order traversal : 2 4 3 6 8 7 5

Tracing :

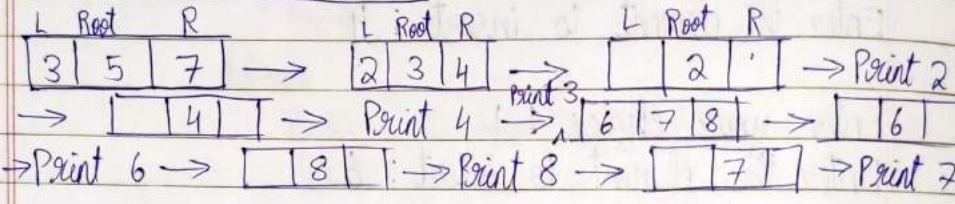
In-Order Traversal :



Pre-Order Traversal:



Post-Order Traversal:



leetcode (Linked List - Rotate):

```
struct ListNode* rotateRight (struct ListNode *head, int k){
    if (head == NULL || k == 0) {
        return head;
    }
    int n = 0;
    struct ListNode* last = head;
    while (last -> next != NULL){
        n++;
        last = last -> next;
    }
    n++;
    last -> next = head;
    int rotate = (n - (k % n));
    for (int i = 0; i < rotate; i++) {
        head = head -> next;
    }
    struct ListNode* ptr1 = head;
    while (ptr1 -> next != head) {
        ptr1 = ptr1 -> next
    }
```

```
    ptr1 -> next = NULL;
    return head;
}
```

Output :

All test cases passed.

22/02/24

Week - 8

1. BFS & DFS methods implementation of tree.

Ans:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100

struct Queue
{
    int front, rear, size;
    unsigned capacity;
    int* array;
};

struct Queue* createQueue(unsigned capacity)
{
    struct Queue* queue = (struct Queue*) malloc(sizeof(struct Queue));
    queue -> capacity = capacity;
    queue -> front = queue -> size = 0;
    queue -> rear = capacity - 1;
    queue -> array = (int*) malloc(queue -> capacity * sizeof(int));
    return queue;
}
```

```
int isEmpty ( struct Queue * queue )
{
    return (queue -> size == 0);
}

void enqueue ( struct Queue* queue , int item )
{
    if ( isFull (queue ))
        return ;
    queue -> rear = (queue -> rear + 1) % queue -> capacity ;
    queue -> array [ queue -> rear ] = item ;
    queue -> size = queue -> size + 1 ;
}

int dequeue ( struct Queue* queue )
{
    if ( isEmpty (queue ))
        return -1 ;
    int item = queue -> array [ queue -> front ] ;
    queue -> front = (queue -> front + 1) % queue -> capacity ;
    queue -> size = queue -> size - 1 ;
    return item ;
}

int isFull ( struct Queue* queue )
{
    return (queue -> size == queue -> capacity ) ;
}

struct Graph
{
    int numVertices ;
    int ** adjMatrix ;
};
```

```
struct Graph* createGraph(int numVertices)
{
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph -> numVertices = numVertices;
    graph -> adjMatrix = (int**) malloc(numVertices * sizeof(int*));
    for (int i = 0; i < numVertices; i++)
        graph -> adjMatrix[i] = (int*) malloc(numVertices * sizeof(int));
    for (int i = 0; i < numVertices; i++)
    {
        for (int j = 0; j < numVertices; j++)
            graph -> adjMatrix[i][j] = 0;
    }
    return graph;
}

void addEdge(struct Graph* graph, int src, int dest)
{
    graph -> adjMatrix[src][dest] = 1;
    graph -> adjMatrix[dest][src] = 1;
}

void BFS(struct Graph* graph, int startVertex)
{
    struct Queue* queue = createQueue(MAX_VERTICES);
    int visited[MAX_VERTICES];
    for (int i = 0; i < MAX_VERTICES; i++)
        visited[i] = 0;
    visited[startVertex] = 1;
    enqueue(queue, startVertex);
    while (!isEmpty(queue))
    {
        int currentVertex = dequeue(queue);
        printf("%d", currentVertex);
```

```
        for (int i = 0; i < graph->numVertices; i++)
        {
            if (graph->adjMatrix[currentVertex][i] == 1
                && !visited[i])
            {
                visited[i] = 1;
                enqueue(queue, i);
            }
        }
    }
    free(queue);
}

void DFSUtil(struct Graph* graph, int vertex, int visited[])
{
    visited[vertex] = 1;
    printf("%d", vertex);
    for(int i = 0; i < graph->numVertices; i++)
    {
        if(graph->adjMatrix[vertex][i] == 1 && !visited[i])
            DFSUtil(graph, i, visited);
    }
}

int isConnected(struct Graph* graph)
{
    int visited[MAX_VERTICES];
    for(int i = 0; i < MAX_VERTICES; i++)
    {
        if(!visited[i])
            return 0;
    }
    return 1;
}
```

```
int main()
{
    struct Graph* graph = createGraph(5);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 4);
    printf("BFS traversal: ");
    BFS(graph, 0);
    printf("\n");
    if (isConnected(graph))
        printf("The graph is connected\n");
    else
        printf("The graph is not connected\n");
    for (int i = 0; i < graph->numVertices; i++)
        free(graph->adjMatrix[i]);
    free(graph->adjMatrix);
    free(graph);
    return 0;
}
```

Output:

```
BFS traversal: 0 1 2 3 4
0 1 3 2 4 The graph is connected.
```

HackerRank (Trees - Swapping Nodes):

```
#include <stdio.h>
#include <stdlib.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, int 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(struct node *root, int inc, int level,
                int height)
{
    struct node *tnode;
    if (! root)
        return;
    if (level > height)
        return;
    if (!(level % inc))
    {
        tnode = root->left;
        root->left = root->right;
        root->right = tnode;
    }
    swap_nodes(root->left, inc, level+1, height);
    swap_nodes(root->right, inc, level+1, height);
}

int tail = 0, head = 0;

void enqueue(struct node **queue, struct node *root)
{
    queue[tail] = root;
    tail++;
}

struct node* dequeue(struct node **queue)
{
```

```
    struct node * temp = queue[head];
    head++;
    return temp;
}

int main()
{
    int nodes_count, i, temp, h, tc_num, index, inc,
        temp1, temp2;
    scanf("%d", &nodes_count);
    struct node *root_perm, *root_temp;
    struct node *q[nodes_count];
    for(i=0; i<nodes_count; i++)
        q[i] = NULL;
    i=0, index=1;
    root_temp = root_perm = create_node(1);
    enqueue(q, root_temp);
    while(index <= 2*nodes_count)
    {
        root_temp = dequeue(q);
        scanf("%d", &temp1);
        if(temp1 == -1){

        }
        else
        {
            root_temp->left = create_node(temp1);
            enqueue(q, root_temp->left);
        }
        if(temp2 == -1){

        }
        else
```

```
        {
            root_temp->right = create_node(temp2);
            enqueue(q, root_temp->right);
        }
        index = index + 2;
    }
    h = height(root_perm);
    scanf("%d", &tc_num);
    while(tc_num)
    {
        scanf("%d", &inc);
        temp = inc;
        swap_nodes(root_perm, inc, 1, h);
        inorder(root_perm);
        printf("\n");
        tc_num--;
    }
    return 0;
}
```

Output :

All test cases passed.

Week - 9

1. Hash Function $H: K \to L$ as $H(K) = K \bmod m$
Remainder method implementation.

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

#define TABLE_SIZE 10
```

```
int hashFunction (int key, int m)
{
    return key % m;
}

void insert (int hashtable[], int key, int m)
{
    int i = 0;
    int hkey = hashFunction (key, m);
    int index;
    do
    {
        index = (hkey + i) % m;
        if (hashtable[index] == -1)
        {
            printf ("Inserted key %d at index %d\n", key, index);
            return;
        }
        i++;
    } while (i < m);
    printf ("Unable to insert key %d. Table is full.\n", key);
}

void search (int hashtable[], int key, int m)
```

```
{
    int i = 0;
    int hkey = hashFunction(key, m);
    int index;
    do
    {
        index = (hkey + i) % m;
        if (hashtable[index] == key)
        {
            printf("Key %d found at index %d\n", key, index);
            return;
        }
        i++;
    } while (i < m);
    printf("Key %d not found in the table\n", key);
}

int main()
{
    int hashtable[TABLE_SIZE];
    int i;
    for (i = 0; i < TABLE_SIZE; i++)
        hashtable[i] = -1;
    insert(hashtable, 1234, TABLE_SIZE);
    insert(hashtable, 5678, TABLE_SIZE);
    insert(hashtable, 9012, TABLE_SIZE);
    insert(hashtable, 2318, TABLE_SIZE);
    search(hashtable, 5678, TABLE_SIZE);
    search(hashtable, 2314, TABLE_SIZE);
    search(hashtable, 9998, TABLE_SIZE);
    return 0;
}
```

29/2/24

Output :

```
Inserted key 1234 at index 4
Inserted key 5678 at index 8
Inserted key 9012 at index 2
Inserted key 2318 at index 9
Key 5678 found at index 8
Key 2318 found at index 9
Key 9999 not found in the table.
```

1/3/24