

R. Abhinav

STD: 3

SEC:

ROLL NO.:

SUB:

PS

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)	22/12	Infix to Postfix Conversion Queue Implementation Circular Queue		{ Sp. t }
4)	11/01	Implementation of linked list Leetcode - Minstack Deletion		
5)	18/01	Deletion in linked list Leetcode - Reverse linked list		{ Sp. t }
6)	25/01	Linked list operations Stack and Queue		{ Sp. t }
7)	1/02	Doubly linked list		
8)	15/02	Binary search Tree Rotate linked list - Leetcode		{ Sp. t }
9)	22/02	BFS DFS Hackerank		{ Sp. t }
10)	29/02	Hashing - Linear Probing		

7/12/23

DS

1)

Enter Username: Abhinav
EnterPwd: 1234
Enter Amount to deposit : 9000
Account created successfully.

Enter Amount to withdraw: 7000
7000 withdrawn successfully!
remaining balance : 2000

2) Enter strings: Abhinav / pratham / ranav
sorted order: Abhinav ranav Pratham

3) Enter ²⁰ Array: 1 2 3, 4 5 6, 7 8 9
Enter Key: 6
Key found at position: ~~1,2~~ 1,2

4) Enter string: "Abhinav"

Enter substring: "hin"

check

substring found at ~~①~~ 2.

G.S.T.
24/12/23

5) Enter array: 4 5 6 1 3 4

Enter key : 3

Key last occurrence 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

Enter key : 3

Key found at 2.

8) Enter array: 4 3 5 1 2 6

Max element is 6, Min element is 1.

21/12/23

Lab - 2

//Program to swap two numbers using pointers
include <stdio.h>

```
void swap(int *a, int *b){  
    int temp = *a;  
    *a = *b;  
    *b = temp;
```

```

d main(){
int a,b;
printf("Read values for a,b: ");
scanf("%d%d", &a,&b);
printf("Value before mapping\n a=%d ; b=%d ", );
swap(&a,&b);
printf("Value after mapping\n a=%d , b=%d " );
}

```

1

and values of a, b: 4, 5

me before mapping

-4
S

the after mapping

14

// Stack implementation

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

#define max 5

int stack[max];
int top = -1;

void push(int a){
    if (top == max - 1) {
        printf("Stack is full");
        return;
    }
    top++;
    stack[top] = a;
}

void pop(){
    if (top == -1) {
        printf("Stack is empty");
        return;
    }
    top--;
    printf("Popped element is %d:", stack[top + 1]);
}
```

```
d display(){
    if (top == -1){
        printf("Stack is empty");
        return;
    }
    for (int i = 0; i <= top; i++) {
        printf("%d ", stack[i]);
    }
}

d main(){
    int choice;
    int a, b;
    while (1) {
        printf("Enter an option: \n1. Push element\n2. Pop element\n3. Display elements");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &a);
                push(a);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
        }
    }
}
```

Output

- Select an option:
- ① 1. Push element
 - 2. Pop element
 - 3. Display elements

choice: 1

Enter value: 2

choice: 1

Enter value: 4

choice: 1

Enter value: 3

choice: 3

2 4 3

choice: 2

Popped element is 3

2 4



// Program to demonstrate dynamic allocation

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

```
int main()
```

```
    int n, *arr, arr2;
```

```
    printf("Read length of both arrays: ");

```

```
    scanf("%d", &n);

```

```
    arr = (int*)malloc(n * sizeof(int));

```

```
    arr2 = (int*)calloc(n, sizeof(int));

```

```
    for (int i=0, i<n, i++) {

```

```
        arr[i] = i+1;

```

```
        arr2[i] = n-i;

```

```
}
```

```
    printf("Array 1 before merging: \n");

```

```
    for (int i=0, i<n, i++) {

```

```
        printf("%d ", arr[i]);

```

```
}
```

```
    arr = realloc(arr, 2*n * sizeof(int));

```

```
    for (int i=n; i<2*n, i++) {

```

```
        arr[i] = arr2[i];

```

```
}
```

```
for(int i=0; i<2^n; i++){
    printf("%d ", arr[i]);
}
free(arr);
}
```

Output:

Read length of both arrays: 5
Array 1 before merging:

1 2 3 4 5

Array 2 after merging:

1 2 3 4 5 6 7 8 9 10

✓
S 1
D 2 3

// Program to convert infix to postfix

```
#include <iostream.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]; postfix[size];
    int j=0;
    for(int i=0; i< strlen(infix); i++){
        if(infix[i] == '(')
            push('(');
        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:

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

Postfix expression is AB*CD*+E -

2) // Evaluating Postfix Expression

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

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

```
#include <ctype.h>
```

```
#define max 20
```

```
int stack[20];
```

```
int top = -1;
```

```
void push(int a){
```

```
    stack[++top] = a;
```

```
}
```

```
int pop(){
```

```
    return stack[top--];
```

```
}
```

~~void main()~~

~~char postfix[max] = "12 * 34 + 5 - ";~~

~~int result = 0, a, b;~~

~~for (int i = 0; i < strlen(postfix); i++) {~~

~~if (isdigit(postfix[i]))~~

~~result = result * 10 + (postfix[i] - '0');~~

~~else~~

```
b = pop();
a = pop();
switch(postfix[i])
{
    case '+':
        push(a+b);
        break;
    case '-':
        push(a-b);
        break;
    case '*':
        push(a*b);
        break;
    case '/':
        push(a/b);
        break;
}
```

```
result = pop();
```

```
printf("%s = %d", postfix, result);
```

}

Output:

Enter postfix expression: 12*34*+5-

Ans 12*34*+5- = 9

3) //Program to implement Queue

```
#include <stdio.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");
```

```
3
void main() {
    int choice(), a;
    while(1) {
        printf("Queue Operations\nInsert\nDelete\nDisplay");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                scanf("%d", &a);
                insert(a);
                display();
                break;
            case 2:
                delete();
                display();
                break;
            case 3:
                display();
                break;
        }
    }
}
```

Output:

Queue Operations

1. Insert
2. Delete
3. Display

choice: 1

Enter Element: 3

Queue: 3

choice: 1

Enter Element: 4

Queue: 3 4

choice: 2

Queue: 4

choice: 3

Queue: 4

S.P.T
28/12/23

1) // Circular Queue

```
#include <csddio.h>
#include <csddib.h>
#define size 5

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

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

void dequeue() {
    if (count == 0)
        printf("Queue empty!");
    f = (f+1)%size;
    count--;
}
```

```
void display() {
    if (count == 0) {
        printf("In Queue empty!");
    }
    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("1. Select choice 1. Enqueue 2. Dequeue\n"
               "3. Display 4. Choice ");
        scanf("%d", &ch);
        switch(ch) {
            case 1:
                printf("\nEnter value to insert: ");
                scanf("%d", &item);
                enqueue(item);
                break;
            case 2:
                dequeue();
                printf("Item popped");
                break;
        }
    }
}
```

case 3:

```
    display();
    break;
    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

choice: 2

~~Select choice~~

Queue = 4

// Linked List

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

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

```
void insertAtHead(struct node **head, int new) {
    struct node *new_node = (struct node *) malloc(sizeof(struct node));
```

```
    new_node->data = new;
    new_node->next = (*head);
```

```
void insertAtMiddle(struct node *prev_node, int new) {
    if (prev_node == NULL) {
```

3

```
        printf("Given previous node cannot be NULL");
        return;
    }
```

```
    struct node *new_node = (struct node *) malloc(sizeof(struct node));
```

```
    new_node->data = new;
    new_node->next = prev_node->next;
    prev_node->next = new_node;
```

```
void insertEnd(struct node** head, int new) {
    struct node *new = (struct node*) malloc(sizeof(struct node));
    struct node **last = **head - ref;
    new->data = new->data;
    if (**head - ref) == NULL {
        **head - ref = new->node;
        return;
    }
    last->next = new->node;
    return;
}

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

of node);

```
int main() {
    struct node *head = NULL;
    int choice = 0, a;
    while (choice != 3) {
        printf("Select an option\n 1. Insert\n 2. Display\n 3. Exit");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                switch (choice) {
                    case 1:
                        insertAtHead(&head, a);
                    case 2:
                        insertInMid(&head, a);
                    case 3:
                        insertAtEnd(&head, a);
                }
            case 2:
                display(head);
        }
    }
}
```

30

3

Output:

Select an option

1. Insert
2. Display
3. Exit

Choice: 1

1. At Start
2. At Middle
3. At End.

choice: 1

Enter value: 5

Linked List: 5

choice: 1

1. At start

2. At middle

3. At end

choice: 2

Enter value: 8

Linked List: 5 8

choice: 2

Linked List: 5 8

Sept
11/11/2024

1) Delete a node from a linked list.

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

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

{}

```
void del_head (struct node **head){
```

```
    if (*head == NULL){
```

```
        printf ("List is empty\n");
```

```
        return;
```

{

```
    struct node *temp = (*head) -> next;
```

```
    free (head);
```

```
    *head = temp;
```



{

```
void del_end (struct node *head){
```

```
    if (head == NULL){
```

```
        printf ("List Empty\n");
```

```
        return;
```

{

```
    while (last -> next != NULL){
```

```
        prev = last;
```

```
        last = last -> next;
```

{

```
    prev -> next = NULL;
```

{

```
void del_mid(struct node *target, struct node *head){  
    struct node *temp = target->next;  
    struct node *prev = head;  
    while (prev->next != target)  
        prev = prev->next;  
    free(target);  
    prev->next = temp;  
}
```

```
void del_val(int val, struct node **head){  
    struct node *temp = *head;  
    if (temp->data == val){  
        del_head(head);  
        return;  
    }  
    while (temp->next != NULL){  
        if (temp->data == val){  
            del_mid(temp, *head);  
            return;  
        }  
        temp = temp->next;  
    }  
}
```

}

```

int main()
{
    struct node * head = NULL;
    for(int i = 1; i <= 5; i++)
        append(&head, i);
    display(head);
    int choice, choice3, val, pos, prval;
    while(1)
    {
        printf("-----\n");
        printf("1. Insert\n2. Delete\n3. Display\nChoice:");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                {
                    printf("Enter value to add");
                    printf("1. Add at head\n2. Add at end");
                    printf("3. Add at middle");
                    choice3 = choice;
                    if (choice3 == 1)
                        del_head(&head);
                    else if (choice3 == 2)
                        del_end(&head);
                    else
                        del_mid(&head);
                }
            case 2:
                {
                    printf("Delete at head\n2. Delete at end");
                    printf("3. Delete at middle");
                    choice3 = choice;
                    if (choice3 == 1)
                        del_head(&head);
                    else if (choice3 == 2)
                        del_end(&head);
                    else
                        del_mid(&head);
                }
        }
    }
}

```

Output:

linked list : 1 2 3 4 5

1. Delete at head

2. Delete at end

3. Delete at middle

choice: 1

linked list : 2 3 4 5

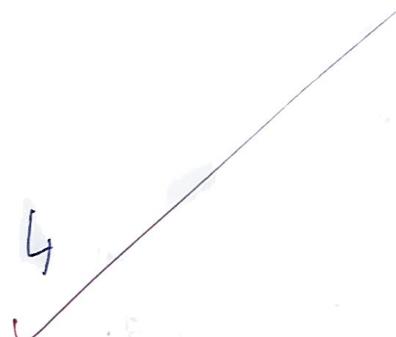
choice: 2

linked list : 2 3 4

choice: 3

Enter Value: 3

linked list : 2 4



minstack

```
edef struct {
    int array[30000];
    int min[30000];
    int top1;
    int top2;
```

MinStack * minStackCreate()

```
MinStack *obj = (MinStack*) malloc (sizeof(MinStack));
obj->top1 = -1;
obj->top2 = -1;
return obj;
```

int 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;
```

}

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];

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

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

void bubbleSort(struct node *head)
{
    struct node *prev;
    struct node *cur;
    int nec, flag = 1, flag1 = 1;
    while (flag)
    {
        prev = head;
        while (prev != NULL && prev->next != NULL)
        {
            cur = prev->next;
            if (cur->data < prev->data)
            {
                nec = cur->data;
                cur->data = prev->data;
                prev->data = nec;
            }
            prev = prev->next;
        }
        flag = flag1;
    }
}
```

```
while (prev != NULL)
{
    if (max > prev->data)
        flag = 0
    max = prev->data;
    prev = prev->next;
}
```

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

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

```
int main()
{
    struct node *head = NULL;
    append(&head, 5);
    append(&head, 2);
    append(&head, 1);
    append(&head, 4);
    display(head);
    bubble-sort(head);
    display(head);
    reverse(head);
    display(head);
    concat(head, head);
    display(head);
    return 0;
}
```

•••
25/1/21

Output:

linked list: 5 2 1 3 4

sorted linked list: 1 2 3 4 5

reversed linked list: 5 4 3 2 1

concat linked list: 5 4 3 2 1 4 5 2

2) // Stack, Implementation
and due.

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

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

```
void append(struct node** head, int newdata){
    struct node* new_node = (struct node*) malloc(sizeof(struct
new_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;
    }
}
```

3

```
void del_end(struct node *head){  
    struct node *last = head;  
    while (last->next != NULL) {  
        prev = last;  
        last = last->next;  
    }  
    free(last);  
    prev->next = NULL;
```

```
int main{  
    int choice=1, a;  
    while(choice<4){  
        printf("1. push\n2. pop\n3. display\n4. choice");  
        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;  
        }  
    }  
}
```

Output:

1.push
2.pop
3.Display
choice: 1
value: 1
choice: 1
value: 2
choice: 3
① stack: 1 2
choice: 2
~~stack~~
choice: 3
stack: 1

3) // Queue implementation

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

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

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

```
printf("%n");
```

```
void del_head(struct node **head){  
    if (*head == NULL)  
        struct node *temp = (*head) -> next;  
    free (*head);  
    *head = temp;
```

```
}
```

```
int main(){  
    struct node *head = NULL;  
    int choice, a;  
    while (choice < 4){  
        printf("1. Push\n2.Pop\n3.Display\n4.choice");  
        scanf("%d", &choice);  
        switch (choice){  
            case 1:  
                // append (& head, a);  
                display (head);  
                break;  
            case 2:  
                // del_head (& head);  
                display (head);  
                break;  
            case 3:  
                display (head);  
        }  
    }  
    return 0;
```

Output:

- 1. Insert
- 2. Delete
- 3. Display

choice: 1

Value: 1

choice: 1

Value: 2

choice: 1

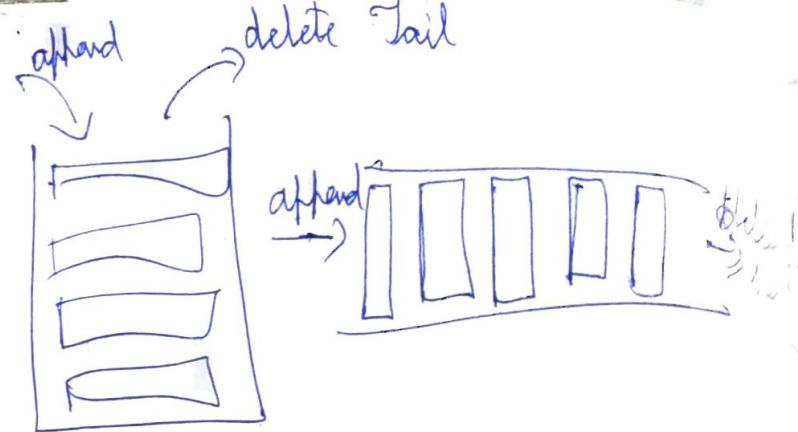
Value: 3

choice: 3

Answer: 1 2 3

choice: 2

Answer: 2 3



1) Leetcode:

```
void addAtHead (struct ListNode** head, int val) {
    struct ListNode* new = (struct ListNode*) malloc
        (sizeof(struct ListNode));
```

$\text{new} \rightarrow \text{val} = \text{Val}$;

$\text{new} \rightarrow \text{next} = \text{NULL}$;

```
if (*head == NULL) {
```

$*\text{head} = \text{new};$

else {

while ($\text{prev} \rightarrow \text{next} \neq \text{NULL}$)

$\text{prev} = \text{prev} \rightarrow \text{next}$

$\text{prev} \rightarrow \text{next} = \text{new}$

}

3

```
int length (struct ListNode* head) {
```

int len = 0;

while ($\text{prev} \neq \text{NULL}$) {

$\text{len}++;$

$\text{prev} = \text{prev} \rightarrow \text{next}$

}

NP
1/2/24

struct ListNode** splitListParts (struct ListNode* head, int k,
int* list[
{

struct ListNode** heads = (struct ListNode**) malloc
(sizeof(struct ListNode*));

for

while (prev != NULL) {

for struct ListNode* i: prev = head;

heads[i] = n: head;

int common = len / k;

int extra = len % k;

int i = 0

while (prev != NULL) {

for (int j = 0; j < common + ((extra > 0) ? 1 : 0);

j++) {

append (&heads[i], prev->val);

prev = prev->next;

}

i++;

extra--;

}

*returnSize = k;

return heads;

```

2) #include <stdio.h>
    #include <stdlib.h>
    #include <stdbool.h>

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

Node * head = NULL;
int count = 0;

void insert(int data, int position){
    if(position == 0){
        Node * newNode = malloc(sizeof(Node));
        newNode->data = data;
        newNode->next = head;
        newNode->prev = NULL;
        if(head != NULL){
            head->prev = newNode;
            head = newNode;
        }
        return;
    }

    if(position == count){
        Node * newNode = malloc(sizeof(Node));
        newNode->data = data;
        newNode->next = NULL;
    }
}

```

```
while (temp->next != NULL) {
    temp = temp->next;
    temp->next = newNode;
    newNode->prev = temp;
    count++;
}
return;
```

```
} else if (position > count || position < 0) {
    printf("Unable to insert at the position.");
    return;
```

```
{ else {
    Node * temp = head;
    for (int i = 0; i < position - 1; i++) {
        temp = temp->next;
    }
    Node * newNode = malloc (sizeof (Node));
    newNode->next = temp->next;
    newNode->prev = temp;
    count++;
}
return;
```

```
}
```

```
}
```

```

void delete (int element) {
    int position = 0;
    Node * temp = head;
    if (head == NULL) {
        printf ("List is empty.");
        return;
    }
    for ( ; position < count; temp = temp->next; position++)
        if (temp->data == element) break;
    if (position == 0) {
        Node * temp = head;
        temp = temp->next;
        temp->prev = NULL;
        free (head);
        head = temp;
        count--;
        return;
    }
    else if (position > count || position < 0) {
        printf ("Unable to delete");
        return;
    }
    else {
        Node * temp = head;
        for (int i=0; i < position; i++)
            temp = temp->next;
    }
}

```

```
temp->next->prev = temp->prev;
temp->prev->next = temp->next;
free(temp);
count--;
return;
}
```

}

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

}

```
int main(){
int data, choice, pos;
printf("1. Insert\n2. Delete\n3. Exit\nChoice:");
scanf("%d", &choice);
while(choice!=3){
    if(choice==1){
        printf("Enter data and position:");
        {
            insert(data, pos);
        }
    }
    else if(choice==2){
        printf("Enter element:");
        delete(pos);
    }
}
}
```

3

```
    display();  
}  
return 0;  
}
```

Output:

- 1. Insert
- 2. Delete
- 3. Exit

choice: 1

Enter: 1 0

Linked list: 1

choice: 1

Enter: 2 1

linked list: 1 2

Enter: 3 2

linked list: 1 2 3

choice: 2

Enter element: 2

linked list: 1 3

Lab-8

15/02/2022

1)

```
//Binary tree  
#include <stdio.h>  
#include <stdlib.h>
```

```
typedef struct BST {  
    int val;  
    struct BST * left;  
    struct BST * right;  
} Node;
```

```
node* newNode(int val){  
    node* newN = (node*) malloc(sizeof(node));  
    newN->val = val;  
    newN->left = NULL;  
    newN->right = NULL;  
    return newN;
```

}

```
void insert(node *root, node *temp){  
    if (root->val > temp->val) {  
        if (root->left != NULL)  
            insert(root->left, temp);  
        else  
            root->left = temp;  
    }
```

```
if (temp->data > root->data) {
    if ((root->right != NULL) &
        insert (root->right, temp));
    else
        root->right = temp;
}
```

```
}
```

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

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

~~12345~~

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

```
void main()
{
    int choice;
    node *root = NULL; *temp;
    printf("1. Create new Insert(12, Display");
    switch (choice)
    {
        case 1:
            printf("Enter data: ");
            scanf("%d", &val);
            if (root == NULL)
                temp = newNode(val);
            root = temp;
            break;
        }
        else
            insert (root, temp);
    }
}
```

```
case 2:  
    printf ("1. Inorder in L.Preorder in 3. Postorder");  
    scanf ("%d", &choice);  
    if (choice == 1)  
        inorder (root);  
    else if (choice == 2)  
        preorder (root);  
    else  
        postorder (root);  
    break;  
}
```

}

)

output:

1. Insert
2. Display

choice: 1

Enter value to insert: 50

choice: 1 - - -

Enter value to insert: 80

choice: 1

Enter value to insert: 60

choice: 1 - - -

Enter value to insert: 20

- - - - -

1. insert
2. display

Choice : 2

preorder: 50 20 10 40 70 80 90

postorder: 10 40 20 20 60 90 70 50

inorder: 10 20 40 50 60 70 90

2) // Leetcode

```
struct ListNode * rotateRight(struct ListNode* head, int k){  
    if (head == NULL)  
        return head;  
    int lengths = 1;  
    struct ListNode *temp1 = head, *temp2;  
  
    while (temp1->next != NULL){  
        temp1 = temp1->next;  
        lengths++;  
    }  
  
    int rotate = lengths - (k % lengths);  
    rotate = rotate - 1;  
    temp1 = head;  
  
    while (rotate != 1){  
        rotate--;  
        temp2 = temp1->next;  
    }  
  
    temp1->next = NULL;  
    head = temp2;  
    temp1->next = head;  
  
    return head;  
}
```

~~NP
(5/2)2u~~

i) BFS.

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

```
#define SIZE 7
```

```
void push(int a);
```

```
int pop();
```

```
void display();
```

```
void bfs(int graph[][SIZE]);
```

```
int thos=-1,xpos=-1;
```

```
int queue[SIZE];
```

```
int main()
```

```
int adj_matrix[SIZE][SIZE] = {
```

$$\{0, 1, 0, 1, 0, 0, 0\},$$

$$\{1, 0, 1, 1, 0, 1, 0\},$$

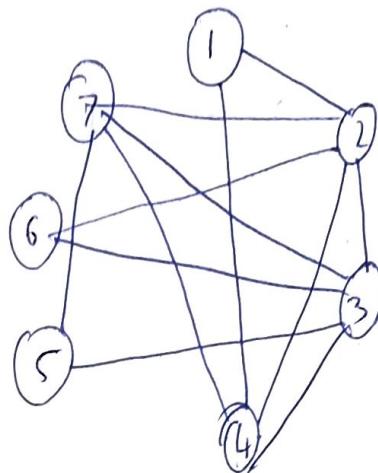
$$\{0, 1, 0, 1, 1, 1, 0\},$$

$$\{1, 1, 1, 0, 0, 0, 1\},$$

$$\{0, 0, 1, 0, 0, 0, 1\},$$

$$\{0, 1, 1, 0, 0, 0, 0\},$$
~~$$\{0, 1, 0, 1, 0, 0, 0\}$$~~

```
}
```



```
for (int i=0; i<SIZE; i++)  
    queue[i] = -1;  
bfs(adj_matrix);
```

```
return 0;
```

```
}
```

```
void bfs(int graph[][size]) {
```

```
    int visited[size];
```

```
    for  
        push(0);
```

```
    visited[0] = 1;
```

```
    while (ffos != size) {
```

```
        for (int i=0; i<size; i++) {
```

```
            if (graph[queue[ffos]][i] == 1 && visited[i] == 0)
```

```
                push(i);
```

```
            visited[i] = 1;
```

```
}
```

```
}
```

```
    printf("%d ", pop());
```

```
}
```

```
}
```

```
void push(int a){  
    if(count < (size-1)){  
        queue[t+ffos]=a;  
        ffos++;  
    }  
    else  
        printf("Queue full");  
}
```

```
int pop(){  
    if(ffos == -1){  
        printf("Queue Underflow");  
    }  
    queue[ffos]=-1;  
    ffos++;  
}
```

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

Output:

BFS: 0 1 3 2 5 6 4

2) DFS:

```
#include <ntddio.h>
#include <ntdll.h>
#include <ntdbooth.h>
```

```
#define size 7
```

```
int top = -1;
```

```
int stack[size];
```

```
void push(int a){
```

```
    if (top == size - 1){
```

```
        printf("Stack Overflow");
```

```
    stack[top + 1] = a;
```

}

~~int~~

```
int pop(){
```

```
    if (top == -1){
```

```
        printf("Stack Underflow");
```

```
    return stack[top--];
```

}



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

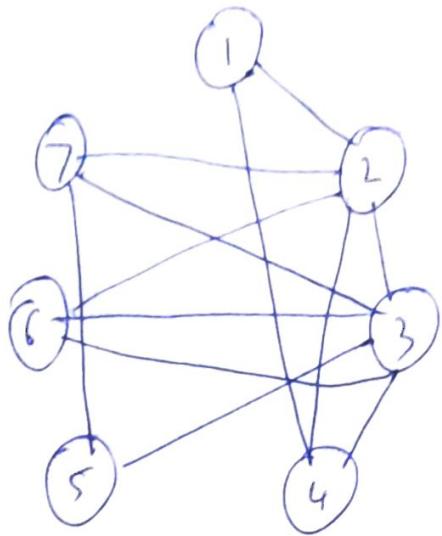
    push(0);
    visited[0] = 1;
    printf("DFS Traversal: 0");

    while (top != -1) {
        bool new_node = false;
        for (int i = 0; i < size; i++) {
            if (graph[stack[top]][i] == 1 && visited[i] == 0) {
                new_node = true;
                push(i);
                visited[i] = 1;
                break;
            }
        }
        if (!new_node)
            pop();
    }
}
```

```

int main(){
    int adj_matrix [n][n] = {
        {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, 1, 1, 0, 0, 0, 0}
    };
    for (int i = 0; i < n; i++)
        stack[i] = -1;
    dfs(adj_matrix);
    return 0;
}

```



Output:

BF

DFS traversal: 0 1 2 3 4 6 5

Sp.1
23/2/20

3) Hackerrank:

```
1 void map_nodes_at_level (struct node *root, int  
                           int level, int height),  
                           struct node *node;  
  
if (!root){  
    return;  
}  
if (level > height)  
    return;  
if (! (level % inc)) {  
    node = root -> left;  
    root -> left = root -> right;  
    root -> right = node;  
}  
map_nodes_at_level (root -> left, inc, level + 1);  
map_nodes_at_level (root -> right, inc, level + 1);
```

{

Sg. 1
22/3/24

Lab - 10

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

```
# define Table-Size 10
```

W

```
int h[Table-Size] = {NULL};
```

```
void insert()
```

```
int key, index, i, flag = 0, hkey;
```

```
printf("\nEnter a value to insert into hash table");
```

```
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("\nElement cannot be inserted\n")
```~~

```
}
```

```
void search() {
 int key, index, i, flag = 0, hkey;
 printf("Enter search: ");
 scanf("%d", &key);
 hkey = key % Table_size;
 if (h[hash] == key)
 {
 printf("Value is found at index %d", index);
 break;
 }
 if (i == Table_size)
 printf("Value is not found");
}
```

```
void main() {
 int opt, i;
 while (1)
 {
 printf("1. Insert 2. Display 3. Exit\n");
 scanf("%d", &opt);
 switch (opt)
 {
 case 1:
 insert();
 case 2:
 display();
 }
 }
}
```

case 3:  
    search();

case 4: exit(0);

}

}

}

Author

prev

- 1. Insert
- 2. Delete
- 3. Search
- 4. Exit

choice: 1

Enter value to insert: 2

choice: 1

Enter value to insert = 6

~~choice: 1~~

~~Enter Value to insert: 3~~

|            | Value |
|------------|-------|
| at index 0 | 0     |
| at index 1 | 0     |
| at index 2 | 2     |
| at index 3 | 3     |
| at index 4 | 0     |
| at index 5 | 0     |
| at index 6 | 6     |
| at index 7 | 0     |
| at index 8 | 0     |

~~876~~  
~~29|2|24~~