7/12/23 DS

1) A
Enter Username: Abhinav
Enter Pwd: 1234
Enter Amount to deposit : 9000
Account created succesfully.

Enter Amount to withdraw: 7000
7000 withdrawed succesfully!
Remaining balance : 2000

2) Enter strings: Abhinav / pratham / pranav
Sorted order : Abhinav Pranav Pratham

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

4) Enter string: "Abhinav"
Enter substring: "hin"
~~check~~
substring found at @ 2.

Sri 24/12/23

5) Enter array: 4 5 6 1 3 4
Enter key: 3
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
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>

d 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 swapping \n a=%d ; b = %d ");
  swap(&a,&b);
  printf("Value after swapping \n a=%d , b = %d");
```

ut:

d values of a,b: 4 5
ue before swapping
4
5

ue after swapping
5
4

2)

```
// 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("\n select an option: \n1. Push element \n2. Pop Elem
        \n3. Display elements");
    scanf("%d", &choice);
    switch(choices){
      case 1:
          printf("Enter value:");
          scanf("%d",&a);
          push(a);
          break;

      case 2:
          pop();
          break;

      case 3;
          display;
  }
}
```

Output:

Select an option:
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;
        arr[i] = n+1-i;
    }

    printf("Array 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] = arr[i-n];
    }
```

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

Output:

Read lengths of both arrays: 5

Array 1 before merging:

1 2 3 4 5

Array 1 after merging:

1 2 3 4 5 6 7 8 9 10

28/12/23

Lab-3

28/12/23

```
// Program to convert infix to postfix
#include <stdio.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:

Enter 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(isalnum(postfix[i]))
            push(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;
         case '^':
                push(a^b);
                break;
        }
    }
 result = pop();
 printf("%s = %d", postfix, result);
}
```

Output:

Enter postfix expression: 12*34*+5-

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]=0
}
}

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, a;
    while(1){
        printf("Queue Operations\n Insert\n Delete\n Display
        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

28/12/23

Lab-4 11/1/24

1) // 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!");
    f = (f+1) % size;
    count--;
}
```

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

nt main(){
    int ch, item;
    while(1){
        printf("\n Select choice \n 1. Enqueue \n 2. Dequeue \n
                3. Display \n Choice:");
        scanf("%d", &ch);

        switch(ch){
            case 1:
                printf("\n Enter value to insert:");
                scanf("%d", &item);
                enqueue(item);
                break;

            case 2:
                dequeue();
                printf("\n 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

~~Enter value~~

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->data = new;
    new->next = (*head);
}

void insertatMiddle(struct node *prev_node, int new){
    if(prev == NULL){
        printf("Given previous nod cannot be NULL");
        return;
    }
    struct node *new = (struct node*) malloc(sizeof(struct node));
    new->data = new_dade;
    new->next = prev->next;
    prev->next = new_node;
}
```

```
void insertAtEnd(struct node **head, int new){
    struct node *new = (struct node*) malloc(sizeof(
    struct node *last = *head_ref;

    new_node -> 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: ");
    while (node != NULL){
        printf("%d ", node -> data);
        node = nod -> next;
    }
    printf("\n");
```

```
of node));
int main(){
    struct node *head =NULL;
    int choice=0, a;
    while(choice!=3){
        printf("select an option\n1. Insert\n2. Display
                                    \n3. Exit");
        scanf("%d",&choice);
        switch(choice){
            case 1:
              switch(choice){
                  case1:
                    insertatHead(&head, a);
                  case2;
                    insertinMid(&head, a);
                  case3:
                  insert at End(&hed, a);
              }
            case 2:
                display(head);
        }
    }
```

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: 3
Enter value: 8
Linked List: 5 8

choice: 2
Linked List: 5 8

11/1/24

Lab - 5

18/1/24

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 = 0; i <= 5; i++)
        append(&head, i);
    display(head);
    int choice, choice3, val, pos, fval;

    while(1)
    {
        printf("----------\n");
        printf("1.Insert\n2.Delete\n3.Display\nChoice:");
        scanf("%d", &choice);

        switch(choice)
        {
            case 2:
                printf("Delete at head\n 2. Delete at end
                    3. Delete at middle");
                scanf("%d", &choice1);
                display(head);
                if(choice==1){
                    del_head(&head);
                }
                else if(choice==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

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

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

id 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;
    }
```

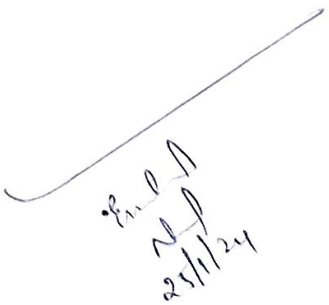
```
void minStackPop(MinStack* obj) {
    obj->top1--;
    obj->top2--;
}

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

int minStackGetMin(MinStack* obj){
    return obj->min[obj->top2];
}
```

Lab 6 25/1/17

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

struct node{
    int data;
    struct node *next;

};

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

```
        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=NUL

    while (current != NULL)
    {
        next = current->next;
        current->next= prev;

        prev = current;
        current = next;
    }
    *head = prev;

}
void concat(struct nod *head1, struct node *head
    struct node *prev= head2;
    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, head2);
    display(head);
    return 0;
}
```



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 2

2) // Stack implementation and push.

```
#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
    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 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\nChoice");
        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;
        }
    }
```

End
25/11/24

Output:

1. Push
2. Pop
3. Display

choice: 1

value: 1

choice: 1

value: 2

choice: 3

1 Stack: 1 2

choice: 2

~~Stack: 1~~

choice: 3

stack: 1

5) //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)->nxt;
        free (*head);
        *head = temp;
}

int main(){
        struct node* head = NULL;
        int choice, a;

        while (choice < 4){
            printf("1. Push\n2.Pop \n3.Display\n Choice");
            scanf("%d", &choice);
            switch(choice){
            case 1:
                push(&head, a);
                display(head);
                break;

NP      case 2:
25/11/22        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

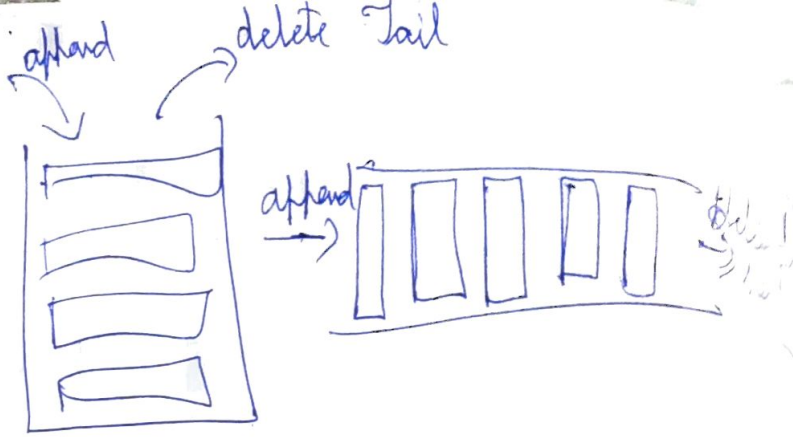
Value : 3

choice : 3

Queue : 1 2 3

choice : 2

Queue : 2 3



Lab-7

1) Leetcode:

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

    new->val = val;
    new->next = NULL;

    if(*head == NULL){
        *head = new;
    }
    else{
        while(prev->next != NULL)
            prev = prev->next

        prev->next = new
    }
}

int length(struct ListNode* head){
    int len=0;
    while(prev != NULL){
        len++;
        prev = prev->next;
    }
}
```

NLP
1/2/24

```
struct ListNode** splitIntoParts(struct ListNode* head, int k,
                                 int* listSize) {
    struct ListNode** heads = (struct ListNode**)malloc
        (sizeof(struct ListNode*));

    for

    while(prev!=NULL){
        for struct ListNode* prev = head;
        heads[i] = nhead;
        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;

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

    }
    else 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 \n 2. Delete\n3. Exit \n Choice");
    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;
    }
```

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

Output:

1. Insert
2. Delete
3. Exit

choice: 1
Enter: 10
Linked list: 1
choice: 1
Enter: 2 1
Linked list: 1 2
choice: 1

Enter: 3 2
Linked list: 1 2 3
choice: 2
Enter element: 2
Linked list: 1 3

Lab-8

15/02/

1)

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

typedef struct BST {
        int val;
        struct BST * left;
        struct BST* Bright;

} 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 -> ~~value~~ > temp -> val) {
            if (root -> data != 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);
    }
}
```

NP
15/2/24

```
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 Insert(n2. Display");
    switch(choice){
        case 1:
          printf("Enter data:");
          scanf("%d", &val);
          if(root == NULL){
              temp = newNode(val);
              root = temp
          }
          else
              insert(root, temp);
          break;
```

```
case 2:
    printf("1.Inorder \n 2.Preorder \n 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

choices: 1
Enter value to insert: 50

choice: 1
Enter value to insert: 20

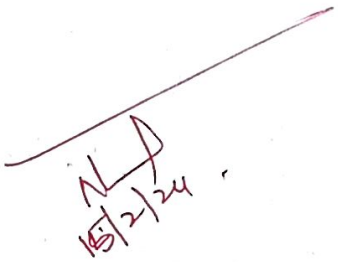
choice: 1
Enter value to insert: 60

choice: 1
Enter value to insert: 20

1. insert
2. display
Choice 2
preorder: 50 20 10 40 70 60 90
postorder: 10 40 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 *temp = head, *temp2;

    while(temp->next != NULL){
        temp = temp->next;
        lengths++;
    }
    int rotate = lengths - (k % lengths);
    rotate = rotate - 1;
    temp = head;
    while(rotate != 1){
        rotate--;
        temp2 = temp->next;
    }
    temp2 = temp->next;
    head = temp2;
    temp->next = NULL;
    return head;
}
```

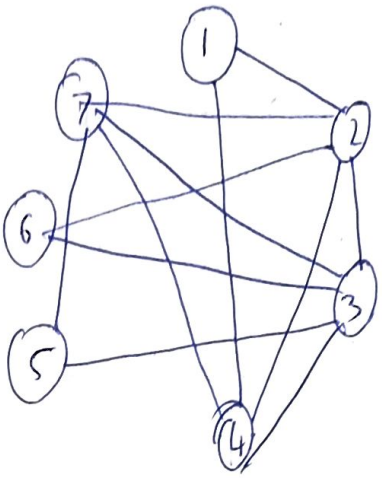


Lab-9

22/2/24

1) BFS

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

#define SIZE 7

void push(int a);
int pop();
void display();
void bfs(int graph[][SIZE]);
int fpos=-1,rpos=-1;
int queue[size];

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



```
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(ftop != size){
        for(int i=0; i<size; i++){
            if(graph[queue[ftop]][i] == 1 && visited[i] == 0)
                push(i);
                visited[i] = 1;

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

}
}
```

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

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

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

Output:

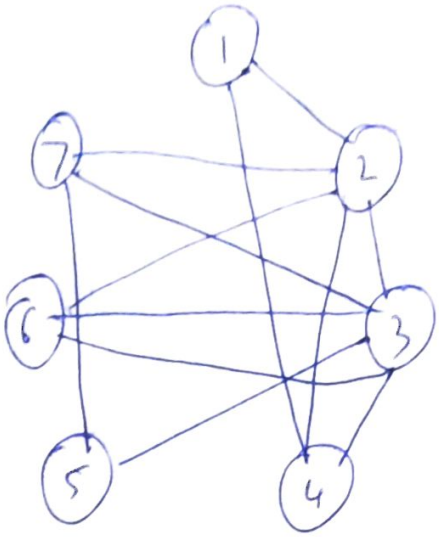
BFS: 0 1 3 2 5 6 4

2) DFS:

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

#define size 7
int top = -1;
int stack[size];

void push(int a){
    if (top == size-1){
        printf("Stack Overflow");
    stack[++top] = a;

}

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]){
                new_node = true;
                push(i);
                visited[i] = 1;
                break;
            }
        }
        if(!new_node)
            pop();
    }
}
```

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



Output:

~~BF~~

DFS traversal: 0 1 2 3 4 6 5

Set 1
23/2/24

3) Hackerrank:

```
void swap_nodes_at_level(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_at_level(root->left, inc, level+1, h
    swap_nodes_at_level(root->right, inc, level+1,
}
```

Sol.1

22/3/24.

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("\n Enter 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("\n Element cannot be inserted\n");
    }
```

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

void main(){
    int opt, i;
    while(1)
    {
        printf("\n Press 1. Insert \t 2. Display \t 3. Search
            \t 4. Exit \n");
        scanf("%d", &opt);
        switch(opt)
        {
            case 1:
                insert();
            case 2:
                display();
```

```
            case 3:
                search();
            case 4: exit(0);
        }
    }
}
```

Output

```
Press
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 >	0
at index 8	0

8th
29/2/24