INDEX

NAME: Pratham STD.: CSE SEC.: 3D ROLL NO.: _____ SUB.: DS LAB

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	21/12/23	pointer wrapping Dynamic memory Stack implement		
2.	28/12/23	Infix to postfix Postfix evaluation		Sri T
3	11/01/24	queue, circular queue		
4	18/01/24	Singly linked list Leetcode		
5	25/01/24	Singly linked list Leet code		
6	1/2/24	Singly linked list: sorting, reversal, concatination ~~Leet code~~ Stack implimentation		Sri T
7.	15/2/24	Doubly linked list: insert, delete Leet code		
8.		Binary tree: traversal, display Leet code		Sri T
9.	22/2/24	Graph Traversal: BFS, DFS		
10.	29-2-24	Hashing		

1 Enter username : Rishi
Enter password : 1321
Enter amount to deposit : 10000
Account created successfully

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5000 withdrawn successfully!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

2. Enter string : ~~Pratham~~ Rishi Shlok Pratham
Sorted order: Pratham Rishi Shlok

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

4. Enter string : "Granpaa"
Enter substring : "np"
substring found at 2

5. Enter array : 4 6 8 7 9 12
Enter key : 9
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

Stack implementation

```
#include <stdio.h>
#include <stdlib.h>
#define size 10
int pos = -1
int array[size];
void push (int a);
int pop ();
void display ();

void main ()
{
printf ("1. Push \n 2. Pop \n. Display stack \n 4. Exit");
printf ("\n Enter choice ");
int choice;
int a;
scanf ("%d", &choice);
while (choice != 4)
{
switch (choice)
{
case 1:
printf ("Enter number to be pushed");
scanf ("%d", &a);
push (a);
break;

case 2:
a = pop ();
printf ("Integer popped = %d\n", a);
break;

case 3:
display ();
break;
}}
```

```
        printf("Enter choice");
        scanf("%d", &choice);
    }
}

void push(int a) {
    if(pos==a)
    {
        printf("Stack overflow");
        return;
    }
    array[++pos] = a;
}

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

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

OUTPUT

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

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

Dynamic memory allocation

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

void main()
{
    int *p, *q;
    int n;
    int i;
    printf("read n");
    scanf("%d", &n);
    p=(int*) malloc(n * sizeof(int));
    printf(" Enter %d Elements ", n);
    for (i=0; i<n; i++){
    {
        scanf("%d", p+i);
    }
    q=(int*) calloc(n, sizeof(int));
    printf("Enter %d Elements ", n);
    for (i=0; i<n; i++)
    { scanf("%d", q+i); }
```

```
p = realloc (p, 7 * sizeof (int));
printf ("Enter seven elements ");
for ( i=0; i<7; i++)
{
    scanf ("%d", p+i);
}
free (p);
free (q);
}
```

OUTPUT

```
Read n  3
Enter 3 Elements 5 4 7
Enter 3 Elements 2 5 4
Enter seven elements 7 8 5 4 1 2 6

Process returned 1 (0x1)  execution time : 21.862s
```

Sol:1
21/2/23

25/12/23

Week-3 : Infix to Postfix

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

#define size 20

void push(char a);
char pop();
void display();
char* postfix(char* exp);
bool character(char c);
bool lower_precedence(char op1, char op2);
bool isEmpty();

int pos = -1;
char Stack[size];
int n;
int main(){
printf("Enter size of expression : ");
scanf("%d", &n);
fflush(stdin);
char* expr=(char*)malloc(n*(n+1)*sizeof(char));
printf("Enter infix expression : ");
scanf("%s", expr);
char* postfix_exp = postfix(expr);
printf("Postfix expression : %s\n", postfix_exp);
}
bool isEmpty()
{
    return pos == -1;
}
```

```
char pop() {
if (pos != -1) {
  printf ("Stack Underflow condition");
        return (char) NULL;
}

  char return_val = stack (pos);
  stack (pos) = (char) NULL;
  pos--;
  return return_val;
}

void display () {
printf ("Stack: ");
for (int i = 0; i < size; i++) {
    printf ("%c ", stack (i));
  }
} printf ("\n");

bool lower_precedence (char op1, char op2) {
  if (op1 == op2 && op2 == '^') return false;

  char op_order [] = {'^', '/', '*', '+', '-'};
  int o1, o2;

  for (int i = 0; i < 5; i++) {
    if (op_order [i] == op1) o1 = i;
    if (op_order [i] == op2) o2 = i;
  }
  return o1 <= o2
}
```

```
char* postfix (char *exp ){
    char* return_exp=(char *)malloc((strlen+3)*sizeof(char));
    int current=0;
    for(int i=0; i<n; i++){
        if (exp[i]=='+'.||exp[i]=='-'||exp[i]=='*'||
            exp[i]=='/'||exp[i]=='^'){
  if(isEmpty) push(exp[i]);
  else if(lower precedence(stack[pos], exp[i])){
      while(lower_precedence(stack[pos], exp(i)]&&
          !isEmpty()){
      if(stack[pos]!='('){ return_exp[current++]=pop();

      else{
      pos--;
      break;
      }
}
push(exp[i]);
}
else push(exp[i]);
}

else if (exp[i]=='(') push('(');
else if(exp[i]==')'){
 while(stack[pos]!='('&& !isEmpty()){
  return_exp(current++) = pop();
}}}
```

Output: Enter no of expression : 9
Enter infix Expression : 1*2+3*4->
Postfix expression : 12* 34*+5-

2 Evaluation postfix expression

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

#define size 20

void push(int a);
int pop();
void display();
bool isEmpty();
bool isOperator(char op);

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

int main() {
    printf("Enter size of expression: ");
    scanf("%d", &n);

    char *exp = (char*) malloc(size * sizeof(char));
    printf("Enter postfix expression on: ");
    scanf("%s", exp);
    int n1, n2;
    for (int i=0; i<n; i++) {
        if (exp[i] >= 0 && exp[i] <= '9') push(exp[i]-'0');
        else if (isOperator(exp[i])) {
            n1 = pop(); n2 = pop();
            switch (exp[i]) {
                case '+':
```

```
push(n2+n1);
    break;
case '-':
    push(n2-n1)
    break;
case '/':
    push(n2/n1);
    break;
case '^':
    push(n1^n2);
    break;
}}}
printf("Final value: %d", pop());

bool isEmpty(){
    return top == -1;

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

int pop(){
    if(top == -1){
    printf("Stack Underflow");
    return (int)NULL;
    }
    int rv
    int return_value = stack[top];
    stack[top] = (int)NULL;
    top--;
    return rv;
```

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

Output : Enter no of expression : 9
Enter postfix expression : 12 * 34 * + 5 -
Final value = 9

Set 1
28/12/23

11-1-24 Week 4

Circular queue:

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

#define size 5

void push (int a) ;
int pop();
void display ();
int fpar =-1, rper = -1; int queue[ size];

int main ()
{
    int choice ;
    printf ("1: Enqueue \n2: ~~Dequeue~~ Dequeue \n3: Display \n4: Exit \nEnter choice");
    scanf ("%d", &choice);
    int a ;
    while (choice != 4)
    {
        switch (choice )
        {
            case 1:
                printf ("Enter integer : ");
                scanf ("%d", &a);
                push(a);
                break ;
            case 2:
                a = pop ();
                printf ("Popped element is : %d\n", a);
                break;
        case 3:
            display ();
            break;
```

```
        default :
            printf ("Wrong input");
        }
        printf ("Enter choice : ");
        scanf ("%d", &choice);
    }
}

void push (int &a)
{
    if ( fpos == -1 && rpos == -1 )
    {
        queue [++rpos] = a;
        fpos ++;
        return;
    }

    else if ((rpos+1) % size == fpos % size)
    {
        printf ("Queue overflow condition");
        return;
    }

    else
    {
        rpos ++;
        queue [(rpos % size)] = a;
        return; } }

int pop ()
{
    if ( fpos == -1 || (fpos % size == rpos % size) )
        printf ("Queue underflow condition"); }
```

```
int n = queue[fpos % size];
queue[fpos % size] = (int) NULL;
fpos--;
return n;
}

void display()
{
    printf("Queue: ");
    for(int i=0; i<5; i++)
        printf("%d", queue[i]);
    printf("\n");
}
```

Output

1: Enqueue
2: Dequeue
3: Display
4: Exit

Enter choice 1
Enter integer: 1
Enter choice: 1
Enter integer: 2
Enter choice: 2
Popped element is: 1
Enter choice: 3
Queue: 02000
Enter choice: 4

Singly linked list:

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

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

node *head = NULL;
int count = 0;

void insert(int data, int position);
void delete(int position);
void display();

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

```
        printf("Enter position : ");
        scanf("%d", &pos);
        delete(pos);
        printf("Count : %d\n", count);
    }
    default();
    printf("Enter choice : ");
    scanf("%d", &choice);
}
return 0;
}

void insert(int data, int position)
{
    if(position == 0)
    {
        node* new_node = (node*) malloc(sizeof(node));
        new_node->data = data;
        new_node->next = NULL;
        head = new_node;
        count++;
        return;
    }}
    else if(position == count)
    {
    node* new_node = malloc(sizeof(node));
    new_node -> data = data;
    new_node -> next = NULL;
    node* temp = head;
    while(temp -> next != NULL)
        temp = temp -> next;
    temp -> next = new_node;
    count++;
    return; }
```

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

else
{
    node* temp = head;
    for( int i=0; i< position-1; i++)
        temp = temp -> next;

    node* new_node = malloc (sizeof(node));
    new_node -> data = data;
    new_node -> next = temp -> next;
    temp -> next = new_node;
    count++;
    return;
    }
}
// display delete
18-1-24 void delete (int position)
{
    if (position == 0)
    {
        node * temp = head;
        head = head -> next;
        free (temp);
        count--;
        return;
    }
    else if (position == count-1)
    {
        node* temp1 = temp -> next;
```

```
temp        temp -> next = NULL;
            free(temp1);
            count--;
            return;
        }
}
// display
void display()
{
    node* temp = head;
    printf("Linked List: ");
    while(temp -> next != NULL)
    {
        printf("%d", temp -> data);
        temp = temp -> next;
    }
    printf("%d", temp -> data);
    printf("\n");
}
```

OUTPUT

```
1. Insert
2. Delete
3. Exit
Choice 1
Enter data and position: 10
Count: 1
Linked List: 1
Enter choice 1
Enter data and position 2 1
Count: 2
Linked List : 1 2
Enter choice 3
```

18/1/24

18/01/24 Week 5

MinStack

```
typedef struct {
    int stack[300000];
    int min;
    int top;
} MinStack;

MinStack* minStackCreate() {
    MinStack* obj = malloc(sizeof(MinStack));
    obj->top = -1;
    obj->min = INT_MAX;
    return obj;
}

void minStackPush(MinStack* obj, int val) {
    if (val <= obj->min) {
        obj->stack[++(obj->top)] = obj->min;
        obj->min = val;
    }
    obj->stack[++(obj->top)] = val;
    return;
}

void minStackPop(MinStack* obj) {
    if (obj->stack[obj->top] == obj->min)
        obj->stack[obj->top] == NULL;
```

```
        obj -> top = 1;
        obj -> min = obj -> stack[(obj -> top)];
    }
    obj -> stack[obj -> top] = NULL;
    obj -> top -= 1;
}
```

OUTPUT [NULL, null, null, null, -3, null, 0, -2]

Reverse Linked List

```
struct ListNode* reverseBetween(struct ListNode* head, int left,
int right) {
    struct ListNode* l = head;
    struct ListNode* r = head;
    int difference = right - left;
    if (left == right)
    {
        return head;
    }
    for (int i = 0; i < left-1; i++)
    {
        l = l -> next;
    }
    for (int i=0; i < right-1; i++)
    {
        r = r -> next;
    }
    printf("%d\n%d\n", l->val, r->val);
    while(difference >= 0)
    {
        int temp = l -> val;
        l -> val = r -> val;
        r -> val = temp;
```

```
    l = l -> next;
    r = l;
    for (int i = 0; i < difference - 2; i++)
    {
        r = r -> next;
    }
    difference -= 2;
}
return head;
}
```

OUTPUT [1,4,3,2,5]

25/1/24

Week 6

1. Singly Linked List operations : sort, reverse and concatenate

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

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

node *head = NULL;
node *head1 = NULL;
int count = 0;

void insert(int data, int position);
void Delete(int position);
void display();
void sort();
void reverse();
void concat(node **  head1, node** head2);

int main()
{
    int data, choice, posi;
    printf("1. Insert\n2. Delete\n3. Exit\n Choice: ");
    scanf("%d", &choice);
    while(choice != 3)
    {
        if(choice == 1)
        {
            printf("Enter data and position: ");
```

```
            scanf("%d %d", &data, &pos);
            insert(data, pos);
            printf("Count: %d\n", count);
        }
        else if(choice == 2)
        {
            printf("Enter position: ");
            scanf("%d", &pos);
            Delete(pos);
            printf("Count: %d\n", count);
        }
        display();
        printf("Enter choice: ");
        scanf("%d", &choice);
    }
    printf("Original Linked List: \n");
    display();
    sort();
    printf("Sorted Linked List: \n");
    display();
    reverse();
    printf("Reversed Linked List: \n");
    display();
    head1 = head;
    head = NULL;
    insert(3, 0);
    insert(4, 1);
    insert(1, 2);
    display();
    concat(&head1, &head);
    head = head1;
    printf("Concatenating with the above linked list gives:\n");
    display();
    return 0;}
```

```
void sort()
{
    int i, j, min_index;
    node *i_node = head, *j_node = head, *min_node = NULL;
    for(int i=0; i<count-1; i++, i_node = i_node->next)
    {
        min_index = i;
        min_node = i_node;
        j_node = i_node->next;
        for(int j = i+1; j<count; j++, j_node = j_node->next)
        {
            if(j_node->data < i_node->data)
            {
                min_index = j;
                min_node = j_node;
            }
        }
        if(min_index != i)
        {
            int temp = i_node->data;
            i_node->data = temp;
        }
    }
}
void reverse()
{
    node* *prev = NULL, *next = NULL;
    while(head != NULL)
    {
        next = head->next;
        head->next = prev
        prev = head;
        head = next; }
    head = prev; }
```

```
void concat(node **head1, node **head2)
{
    node *temp1 = *head1;
    while (temp1->next != NULL)
    {
        temp1 = temp1->next;
    }
    temp1->next = *head2;
    printf
}
```

OUTPUT

Original Linked List:
~~Linked List~~ 2 1 4 3 5
Sorted Linked List:
1 2 3 4 5
Reversed Linked List:
5 4 3 2 1
3 4 1
Concatinating with the above linked list gives:
5 4 3 2 1 3 4 1

Stacks using Linked List:

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

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

node* head=NULL;
int count=0;

void insert(int data);
int delete();
void display();

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

```
            printf("Integer popped = %d\n", delete());
            printf("Count: %d\n", count);
        }
        display();
        printf("Enter choice: ");
        scanf("%d", &choice);
    }
    return 0;
}

void insert(int data)
{
    node* new_node = (node*)malloc(sizeof(node));
    new_node->data = data;
    new_node->next = head;
    head = new_node;
    count++;
    return;
}

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

OUTPUT:

1. Insert
2. Delete
3. Exit

Choice : 1
Enter data : 1
Count : 1
Stack : 1
Enter choice 1
Enter data : 3
Count : 2
Stack : 3 1
Enter choice 2
Integer popped = 3
Count : 1
Stack : 1
Enter choice : 3

Sept
01/2/24

1/2/24

Week 7:

1 W# P to implement doubly link list with primitive operations:

(a) Create doubly linked list
(b) Insert new node to the left of the node
(c) Delete the node based on a specific value
Display contents

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

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

node* head = NULL
int count = 0;

void insert (int data, int position);
void delete (int element);
void display ();

int main() {
        int data, choice, pos;
        printf("1. Insert \n2. Delete \n3. Exit \n Choice: ");
        scanf("%d", &choice);
        while (choice != 3) {
          if (choice == 1) {
                printf("Enter data and position: ");
                scanf("%d%d", &data, &pos);
```

```
            insert(data, pos);
            printf("Count : %d\n", count);
        }
        display();
        printf("Enter choice : ");
        scanf("%d", &choice);
    }
    return 0;
}

void insert(int data, int position){{
    if (position == 0){
        node* new_node = malloc(sizeof(node));
        new_node -> data = data;
        new_node -> next = head;
        new_node -> prev = NULL;
        if (head != NULL) head -> prev = new_node;
        head = new_node;
        count++;
        return;
    }
    else if (position == count){
        node* new_node = malloc(sizeof(node));
        new_node -> data = data;
        new_node -> next = NULL;
        node* temp = head;
        while (temp -> next != NULL)
            temp = temp -> next;
        temp -> next = new_node;
        new_node -> prev = temp;
        count ++;
        return;
    }
    else if (position > count || position < 0){{
```

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

void delete(int element){
    int position=0; node *temp=head;
    if(head == NULL){
        printf("List is empty, cannot delete"); return;
    }
    for(; position<count; temp=temp->next, position++)
        if(temp->data == element) break;

    if(temp == NULL){
        printf("Element does not exist in list");
        return;
    }
    if(position == 0){
        node* temp = head;
        temp = temp->next;
        temp->prev = NULL;
```

```
        free (temp->head);
        head = temp;
        count--;
        return;
    }
    else if (position == count-1) {
        node* temp = head;
        temp = temp -> next;
        temp -> prev = NULL;
        for(int i = 1; i < count-1; i++)
            temp = temp -> next;
        node* temp1 = temp -> next;
        temp -> next = NULL;
        free (temp1);
        count--;
        return;
    }
    elseif (position > count || position < 0){
        printf("Unable to delete at position \n");
        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("\nLinked List: ");
    while(temp -> next != NULL){
        printf("%d", temp -> data);
        temp = temp -> next;
    }
    printf("%d ", temp -> data);
    printf("\n");
}
```

OUTPUT

```
1. Insert
2. Delete
3. Exit
Choice: 1
Enter data and position: 5 0
Count: 1
Linked List: 5
Enter choice 1
Enter data and position: 4 1
Count: 2
Linked List: 5 4
Enter data and position: 6 2
Count: 3
Linked List: 5 4 6
Enter choice 1
Enter data and position: 9 3
Count: 4
Linked List: 5 4 6 9
Enter choice: 2
Enter element: 6
Count: 3
Linked List: 5 4 9
Enter choice: 3
```

Split linked list:

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

OUTPUT

~~head = [1,2,3]~~
~~k = 5~~

head = [1,2,3,4,5,6,7,8,9,10]
k = 3

5/2/24 Week8: Binary tree

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

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

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

int main() {
{
    int data
    insert (&root, 100);
    insert (&root, 20);
    insert (&root, 200);
    insert (&root, 10);
    insert (&root, 30);
    insert (&root, 150);
    insert (&root, 300);

    preorder (&root);
    printf ("\n");

    inorder (&root);
    printf ("\n");

    postorder (&root);
```

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

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

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

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

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

OUTPUT

100	20	10	30	200	150	300
10	20	30	100	150	200	300
10	30	20	150	300	200	100

Trace:

Preorder

100 → preorder(left) → 20 → preorder(left) → 10 → preorder(left) → NULL → preorder(right) → NULL → preorder(right) → 30 → preorder(left) → NULL → preorder(right) → NULL → preorder(right) → 200 → preorder(left) → 100 → preorder(right) → 300 → preorder(left) → NULL → preorder(right) → NULL

Post order:

preorder(left) → preorder(left) → preorder(left) → NULL → preorder(right) → NULL → 10 → preorder(right) → preorder(left) → NULL → preorder(right) → NULL → 30 → 20 → preorder(right) → preorder(left) → preorder(left) → NULL → preorder(right) → NULL → 150 → preorder(right) → preorder(right) → NULL → preorder(right) → NULL → 300 → 200 → 100

Inorder:

inorder(left) → inorder(left) → inorder(left) → NULL → 10 → inorder(right)

Rotate List:

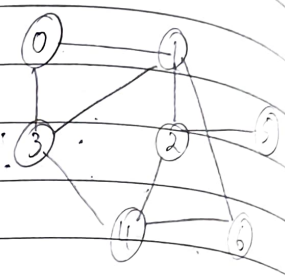
```
struct ListNode* rotateRight(struct ListNode* head, int k)
{
    struct ListNode *temp = head;
    if (head == NULL) return NULL;
    if (head->next == NULL) return head;
    if (k == 0) return head;
    int size = 1;
    for(; temp->next != NULL; temp = temp->next, size++);
    k %= size;
    if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

22-2-24 Week 9: DFS

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

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

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

int main() {
    int adj_matrix[7][7] = {
        {0,1,0,0,1,0,0},
        {1,0,1,1,0,0,1},
        {0,1,0,1,1,1,0},
        {1,1,1,0,0,0,0},
        {0,0,1,0,0,0,1},
        {0,1,1,0,0,0,0},
        {0,1,0,0,1,0,0}
    };
    for(int i=0; i<7; i++) stack[i]=NULL;
    dfs(adj_matrix);
    return 0;
}

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

```
bool new_node = false;
for (int i = 0; i < 7; i++) {
    if (graph[stack[pos]][i] == 1 && visited[i] != 1)
    {
        new_node = true;
        push(i);
        visited[i] = 1; printf("%d", i);
        break;
    }
}
if (! new_node) pop();
}
}

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

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



OUTPUT 0 1 2 3 4 6 5

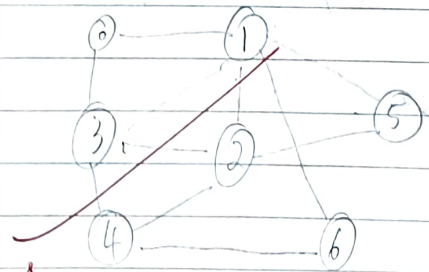
BFS:

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

#define size 7

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

int fpos = -1, rpos = -1;
int queue[size];

int main() {
    int adj_matrix[97][7] = {
{0,1,0,1,0,0,0},{1,0,1,1,0,1,1},{0,1,0,1,1,1,0},
{1,1,1,0,0,0,0},{0,0,1,0,0,0,1},{0,1,1,0,0,0,0},
{0,1,0,0,1,0,0},};
    for(int i=0; i<7; i++) queue[i]=NULL;
    bfs(adj_matrix);
```

```
return 0; }

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

void push (int a)
{
    if (fpos == -1 && rpos == -1)
    {
        queue [++rpos] = a;
        fpos++;
        return;
    }

    else if (rpos == size-1) {
        printf("Queue overflow condition \n");
        return;
    }
    else{
        queue [++rpos] = a;
        return;}
```

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

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

OUTPUT 0 1 3 2 5 6 4



Sept 1
22/2/24

29/2/24

Hashing using linear probing:

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

#define size 10

int table[size];

void push(int data);
int pop(int data);
void search(int data);
void display();

int main() {
{
    for(int i=0; i<size; i++) table[i] = -1;
    int choice;
    printf("1. Insert \n2. Delete \n3. Display \n4. Exit \n
            Choice: ");
    scanf("%d", &choice);
    int a;
    while(choice != 4)
    {
        switch(choice){
        case 1:
            printf("Enter integer to be pushed: ");
            scanf("%d", &a);
            push(a);
            break;
        case 2:
            printf("Enter integer to be popped: ");
            scanf("%d", &a);
```

```
            int res = pop(a);
            if (res == 0) printf("Integer popped\n");
            else printf("Integer not found");
            break;
        case 3:
            display();
            break;

        default:
            printf("IDK");
            break;
    }
    printf("Enter choice: ");
    scanf("%d", &choice);
  }
}

void push(int data) {
    int hash = data % size;
    while(table[hash] != -1 && hash <= (hash+size-1))
    hash = (hash+1) % size;

    if (table[hash] == -1) table[hash] = data;
    else printf("Table is full");
}

int pop(int data)
{
    int hash = data % size;
    while (table[hash] != -1 && hash <= (hash+size-1))
    hash = (hash+1) % size;
    if (table[hash] == data) {
        table[hash] = -1; return 0;
    }
```

```
    return -1;
}

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

Output:

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

Choice: 1
Enter integer to be pushed: 1
Enter choice: 1
Enter integer to be pushed: 3
Enter choice: 3
Table: -1 1 -1 3 13 -1 -1 -1 -1 -1
Enter choice: 4