# INDEX

NAME: Pranav A Rao STD.: ____ SEC.: ____ ROLL NO.: ____ SUB.: Data Structures Obs

| S. No. | Date | Title | Page No. | Teacher's Sign / Remarks |
|---|---|---|---|---|
| 1 | 07/12 | Practice Programs | 01 | 7/12 (10) |
| 2. | 21/12 | Swapping two variables | 03 | |
| | | Dynamic Memory Allocation | 04 | 21/12 (10) |
| | | Stack Implementation | 07 | |
| 3. | 28/12 | Infix to Postfix Conversion | 10 | |
| | | Postfix Evaluation | 13 | 11/1/24 (10) |
| | | Queue implementation | 16 | |
| | | Circular queue implementation | 19 | |
| 4 | 11/01 | Implementation of Linked List | 22 | 18/1/24 (10) |
| | | LeetCode - MinStack | 30 | |
| 5 | 18/01 | Deletion in Linked List | 27 | 18/1/24 |
| 6 | 25/01 | Linkedlist Operations | 33 | (10) |
| | | Stack & Queue in Linked List | 37 | 28/1/24 |
| 7. | 01/02 | Doubly Linked List | 43 | 1/2/24 (10) |
| 8. | 15/02 | Binary Search Tree | 48 | |
| | | LinkedList - Rotate List | 52 | 22/2/24 (10) |
| 9 | 22/02 | BFS | 54 | |
| | | DFS | 56 | 29/2/24 (10) |
| | | HackerRank | 61 | |
| 10 | 29/02 | Hashing - Linear Probing | 58 | |
| | | | | (10) 29/2/24. |

# Practice Programs.

1. Enter
1 to make a new account
2 to withdraw
3 to deposit
4 to check balance
5 to Exit
1
Enter account name
Pranav
Enter age
19
3
Enter amount to deposit
1000
2
Enter amount to withdraw
600
4.
Balance is 400
5

2. Enter element to search
3

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

3 is present

4. Enter string
My name is Pranav
Enter substring
name
Substring is present

5. Enter elements
12 22 33 44 58 69 77 84 91 22
Enter element to search
22
Last occurance of 22 is at 10

6. Enter elements
12 22 33 44 58 69 77 84 91 103
Enter element to search
22
Element 22 is at 2

7. Enter elements
12 22 33 44 58 69 77 84 91 103
Enter element to search
22
Element 22 is at 2

8. Enter number of element: 5
10 20 30 40 50
Biggest is 50
Smallest is 10

Week 1 (21/12/23)

1 Swap two variables

```
#include<stdio.h>

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

void main(){
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d%d", &a, &b);
    printf("Before swapping, a = %d and b = %d \n", a, b);
    swap(&a, &b);
    printf("After swapping, a = %d and b = %d", a, b);
}
```

Output:

```
Enter two numbers: 20 30
Before swapping, a = 20 and b = 30
After swapping, a = 30 and b = 20
```

classmate
Date
Page

## 2 Dynamic Memory Allocation

```
#include <stdio.h>

int main(){
    int *ptr;
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    ptr = (int*) malloc(n * sizeof(int));
    if(ptr == NULL){
        printf("Memory not allocated \n");
        exit(0);
    }

    printf("Memory successfully allocated using malloc\n");
    printf("Enter elements of array \n");
    for(int i=0; i<n; i++){
        scanf("%d", &ptr[i]);
    }

    printf("Elements of array are: ");
    for(int i=0; i<n; i++){
        printf("%d, ", ptr[i]);
    }
    free(ptr);
    printf("Memory freed \n");
    ptr = (int*) calloc(n, sizeof(int));
    if(ptr == NULL){
        printf("Memory is not allocated \n");
        exit(0);
    }
    printf("Memory allocated using calloc\n");
```

```
printf("Enter elements of array \n");
for(int i=0; i<n; i++){
    scanf("%d", &ptr[i]);
}
printf("The elements of the array:\n");
for(int i=0; i<n; i++){
    printf("%d, ", ptr[i]);
}
int n1;
printf("Enter new size for realloc\n");
scanf("%d", &n1);
ptr = (int *) realloc(ptr, n1*sizeof(int));
printf("Memory successfully reallocated \n");
printf("Enter more variables: \n");
for(int i=n; i<n1; i++){
    scanf("%d", &ptr[i]);
}
printf("The elements are ");
for(int i=0; i<n1; i++){
    printf("%d; ", ptr[i]);
}
free(ptr);
printf("Memory freed \n");
}
```

Output:

```
Enter number of elements: 3
Memory successfully allocated using malloc.
Enter elements of array
1
2
3
The elements of the array are 1, 2, 3.
Memory freed.
Memory successfully allocated using calloc.
Enter elements of array:
1
2
3
The elements of the array are 1, 2, 3
Enter new size for realloc: 5
Memmory successfully reallocated
Enter more variables:
4
5
The elements of the array are 1, 2, 3, 4, 5
Memory freed.
```

3 Stack Implementation

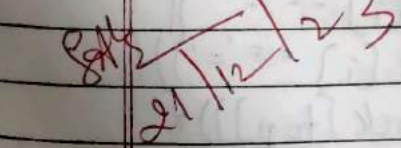
```
#include<stdio.h>

# define size 10

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

void pop();
void push();
void display();

void main(){
   while(1){
      int choice;
      printf("Enter 1 for push \n");
      printf("Enter 2 for pop \n");
      printf("Enter 3 for display \n");
      printf("Enter choice:");
      scanf("%d", &choice);
      switch(choice){
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            default: exit(0)
      }
   }
}
```

```
void push () {
  int x;
  if (top == size - 1){
     printf(" Overflow \n ");
  }
  else {
     printf (" Enter element: ");
     scanf (" %d ", &x);
      top = top + 1;
      stack [top] = x;
   }
}

void pop(){
  if (top == -1) {
     printf (" Underflow ");
  }
  else {
     printf (" Popped element = %d ", stack [top]);
     top = top - 1;
  }
}

void display (){
  if (top == -1) {
    printf (" Underflow");
  }
  else {
     printf (" Elements are : \n ");
     for (int i = 0; i <= n; i++)
```

```
        for (int i = top; i >= 0; i--) {
            printf ("%d \n", stack[i]);
        }
    }
}
```

Output:

```
Enter 1 for push
Enter 2 for pop
Enter 3 for display
Enter choice: 1
Enter element: 1
Enter choice: 1
Enter element: 2
Enter choice: 1
Enter element: 3
Enter choice: 2
Popped element: 3
Enter choice: 3
Enter Elements are:
2
1
Enter choice: 0
Program exited.
```



Week 2 - 08/12/2023

1 Infix to Postfix conversion

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

#define max 20

void push(char a);
char pop();
char stack[max], top = -1;
int pre(char a);

void main(){
    char infix[max], a;
    char post[max];
    printf("Enter infix expression: ");
    scanf("%s", infix);
    int j = 0;
    push('(');
    for(int i = 0; i < strlen(infix); i++){
        if(isalnum(infix[i])){
            post[j] = infix[i];
            j++;
        }
        else if((infix[i] == '+' || infix[i] == '-' ||
                 infix[i] == '/' || infix[i] == '='){
            if(pre(infix[i]) > pre(stack[top])){
                push(infix[i]);
            }
```

```
        else if (pre(infix[i]) <= pre(stack[top])){
            while (1){
                a = pop();
                if (a == '(') {
                    push(a);
                    break;
                }
                post[j] = a;
                j++;
            }
        }
    }
}
while (top != -1) {
    char y = pop();
    if (y == '(') {
        break;
    }
    post[j] = y;
    j++;
}
post[j] = '\0';
printf("%s", post);
}

void push(char a){
    if (top > max-1){
        printf("Stack Overflow");
        exit(0);
    }
    else{
```

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

char pop(){
    if(top==-1){
        printf("Stack underflow");
        exit(0);
    }
    else{
        return stack[top--];
    }
}

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

Output:

Enter infix expression:

2+7*(5%4)-6

Postfix:

2754%*+6-

2. Evaluation of postfix expression

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

# define max 20

void push(float a);
int pop();
int stack[max], top = -1;
float compute(int a, int b, char c);

void main(){
    char postfix[max];
    int a, b;
    char c, d;
```

```
printf("Enter postfix expression: ");
scanf("%s", postfix);
int d, i=0;
for(int i=0; i<strlen(postfix); i++){
    d = postfix[i];
    if(isdigit(d)){
        push(d-'0');
    }
    else if((postfix[i]=='+' || postfix[i]=='-'
        || postfix[i]=='/' || postfix[i]=='*')){
        a = pop();
        b = pop();
        c = compute(a, b, postfix[i]);
        push(c);
    }
}
printf("%d", pop());
}

void push(float a){
    if(top >= max-1){
        printf("Stack Overflow");
    }
    else{
        ++top;
        stack[top] = a;
    }
}
```

```
int pop(){
  if (top == -1){
    printf("Stack underflow");
  }
  else {
    return stack[top--];
  }
}

float compute(int a, int b, char c){
  if (c == '+')
    return a+b;
  else if (c == '*')
    return a*b;
  else if (c == '-')
    return b-a;
  else if (c == '/')
    return b/a;
}
```

Output:

Enter postfix:

345 * 6 / 9 % +

Result: 6

3a Queue Implementation

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

#define max 20

int queue[max];
int front = -1, rear = -1;

void enqueue(int a);
int dequeue();
void display();

void main(){
    int n, m;
    printf("Enter \n 1. to insert, \n 2. to remove, \n
            3. to display \n");
    scanf("%d", &n);
    while(n != 3){
        switch(n){
            case 1: printf("Enter element: ");
                    scanf("%d", &m);
                    enqueue(m);
                    break;
            case 2: m = dequeue;
                    printf("Element removed is %d", m);
                    break;
            default: printf("Invalid Input");
        }
        scanf("%d", &n);
```

```
        printf("\n");
    }
    if(n==3){
        display();
    }
}

void enqueue(int a){
    if(rear == max-1){
        printf("Queue full");
    }
    rear = -1;
    queue[rear] = a;
}

int dequeue(){
    if(rear==-1 || front==rear){
        printf("Queue empty");
        exit(0);
    }
    else{
        front+=1;
        return queue[front];
    }
}

void display(){
    printf("Queue:");
    for(int i=front+1; i<=rear; i++){
        printf("%d\t", queue[i]);
    }
}
```

Output:

```
1. Insert element
2. Delete element
3. Exit
1
Enter element: 23

1. Insert Element
2. Delete element
3. Exit
1
Enter element: 54

1. Insert element
2. Delete element
3. Exit
2
Deleted element: 23

1. Insert element
2. Delete element
3. Exit
1
Enter element: 65
Queue is full
```

8th 11/9/24.

3b. Circular Queue Implementation:

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

#define max 6

int cq[max];
int front = -1, rear = -1;

bool is_full(){
    return (rear+1) % max == front;
}

bool is_empty(){
    return front == -1 && rear == -1;
}

void enqueue(int item){
    if (is_full()){
        printf("Queue is full");
        return;
    }
    if (is_empty()){
        front = rear = 0;
    } else {
        rear = (rear+1) % max;
    }
    cq[rear] = item;
    printf("Enqueued");
}
```

```
int dequeue(){
  if(is_empty()){
    printf("Underflow");
    return -1;
  }
  int deletedItem = cq[front];
  if(front == rear){
    front = rear = -1;
  }
  else{
    front = (front+1) % max;
  }
  printf("Dequeued");
  return deletedItem;
}

int main(){
  int n, ele;
  do{
    printf("\n1. Insert \n2. Delete \n3. Exist:");
    scanf("%d", &n);
    switch(n){
      case 1: printf("Enter element");
              scanf("%d", &ele);
              enqueue(ele);
              break;
      case 2: {
              int deletedItem = dequeue();
              if(deletedItem != -1){
                printf("The element is removed\n");
              }
```

```
        }
        break;
    case 3:
        printf("Exited");
        break;
    default: printf("Invalid option");
    }
}while(n!=3);
return 0;
}
```

Output:

```
1. Insert
2. Delete
3. Exit
1
Enter element: 23

1 Insert
2. Delete
3. Exit
2
Dequeued element: 23
```

Week 3 - 11/01/24

1. Create a singly linked list with following operation
a) Create a linked list
b) Insertion of a node at first position, at any position and at end of list
c) Display the contents of the linked list.

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

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

Node * head = NULL;

void push();
void append();
void insert();
void display();

int main() {
    int choice;
    while (1) {
        printf("1. Insert at beginning \n");
        printf("2. Insert at end \n");
        printf("3. Insert at position \n");
        printf("4. Display \n");
        printf("Enter choice : ");
```

```
scanf("%d", &choice);
switch (choice){
    case 1: push();
            break;
    case 2: append();
            break;
    case 3: insert();
            break;
    case 4: display();
            break;
    default: printf("Exiting the program");
             return 0;
        }
    }
}

void push(){
    Node * temp = (Node *) malloc(sizeof(Node));
    int new_data;
    printf("Enter data in the new node: ");
    scanf("%d", &new_data);
    temp -> data = new_data;
    temp -> next = head;
    head = temp;
}

void append(){
    Node * temp = (Node *) malloc(sizeof(Node));
    int new_data;
    printf("Enter data in the new node: ");
    scanf("%d", &new_data);
```

```
    temp -> data = new_data;
    temp -> next = NULL;
    if (head == NULL){
        head = temp;
        return;
    }
    Node * temp1 = head;
    while (temp1 -> next != NULL){
        temp1 = temp1 -> next;
    }
    temp1 -> next = temp;
}

void insert (){
    Node* temp = (Node*) malloc (sizeof (Node));
    int new_data, pos;
    printf ("Enter data in the new node : ");
    scanf ("%d", &pos);
    temp -> data = new_data;
    temp -> next = NULL;
    if (pos == 0) {
        temp -> next = head;
        head = temp;
        return;
    }
    Node* temp2 = temp1 -> next;
    temp -> next = temp2;
    temp1 -> next = temp;
}
```

```
void display(){
    Node* temp1 = head;
    while (temp1 != NULL){
        printf("%d → ", temp1→data);
        temp1 = temp1→next;
    }
    printf("NULL");
}
```

Output:

```
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5 Exit
Enter choice: 1
Enter data in the new node: 5

Enter choice: 2
Enter data in the new node: 6

Enter choice: 3
Enter data in the new node: 1
Enter position of the new node: 1

Enter choice: 4
5 → 6 → 1 → NULL

Enter choice: 5
Program exited.
```

Week 4 - 18/01/24

1. Create a singly linked list with following operations
a) Create a linked list
b) Deletion of first element, specified element and last element in the list
c) Display the contents of the linked list

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

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

Node* head = NULL;

void pop();
void end_delete();
void delete_at_pos();
void display();
void append();

int main() {
    int choice;
    while (1) {
        printf("1. Insert at end \n");
        printf("2. Delete from beginning \n");
        printf("3. Delete from end \n");
        printf("4. Delete ~~fror~~ at position \n");
```

```
printf("5. Display \n");
printf("6. Exit \n");
printf("Enter choice: ");
scanf("%d", &choice);
switch (choice) {
    case 1:
        append();
        break;
    case 2:
        pop();
        break;
    case 3:
        end_delete();
        break;
    case 4:
        delete_at_pos(); break;
    case 5:
        display();
        break;
    default:
        printf("Exiting the program");
        return 0;
}
}
}

void pop(){
  if(head == NULL){
     printf("Linked list is empty\n");
     return;
  }
```

```
    head = head -> next;
}

void end_delete(){
    if (head == NULL) {
        printf("Linked list is empty");
        return;
    }
    if (head -> next == NULL){
        head = NULL;
        return;
    }
    Node* temp = head;
    Node* temp1 = head;
    temp = temp -> next;
    while (temp -> next != NULL){
        temp = temp -> next;
        temp1 = temp1 -> next;
    }
    temp1 -> next = NULL;
}

void delete_at_pos(){
    int pos;
    printf("Enter position: ");
    scanf("%d", &pos);
    Node* temp = head;
    while (--pos){
        temp = temp -> next;
        if (temp -> next == NULL){
            printf("Not enough elements");
```

```
        }
    }
    Node * temp1 = temp;
    temp1 = temp1 -> next;
    if (temp1 -> next == NULL) {
        printf ("Not enough elements");
    }
    temp1 = temp1 -> next;
    temp -> next = temp1;
}
```

O/P :

1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete at end
5. Display
6. Exit

Enter choice : 1, 1, 1, 2, 1, 3, 1, 4
Enter choice : 2
Enter choice : 3
Enter choice : 4
Enter position : 1
Enter choice : 5

2 → NULL

Week 3- Leetcode

Q. Design a stack that supports push, pop, top and retrieving minimum element in constant time.

```
#include <stdio.h>
#include <stdlib.h>
# define max 4

typedef struct {
    int top;
    int st[max];
    int min[max];
} MinStack

MinStack* minStackCreate(){
    MinStack* stack = (MinStack*)malloc(sizeof(MinSt
    stack -> top = -1;
    return stack;
}


void minStackPush(MinStack* obj, int val){
    if(obj -> top == max - 1){
        printf("Stack Full \n");
        return;
    }
    obj -> st[++obj -> top] = val;
    if(obj -> top > 0){
        if(obj -> min[obj -> top - 1] < val)
            obj -> min[obj -> top] = obj -> min[obj -> top-1]
        else
```

```
        obj->min[obj->top] = val;
    }
    else
        obj->min[obj->top] = val;
}

void minStackPop(MinStack * obj){
    if(obj->top == -1){
        printf("Stack empty\n");
        return;
    }
    else{
        obj->top--;
    }
}

int minStackTop(MinStack* obj){
    if(obj->top == -1){
        printf("Stack empty\n");
        return -1;
    }
    return obj->st[obj->top];
}

int minStackGetMin(MinStack* obj){
    if(obj->top == -1){
        printf("Stack empty\n");
        return -1;
    }
    return obj->min[obj->top];
}
```

O/P:

Input:

["MinStack", "push", "push", "push", "getMin", "pop", "getMin"]

[[], [-2], [8], [-3], [], [], [], []]

Output:

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

18/11/24

25-01-2024

Q WAP to implement singly linked list with sorting reversing and concatenation of two linked list

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

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

~~void InsertAtEnd~~

struct Node* head1 = NULL;
struct Node* head2 = NULL;

void insertAtEnd(struct Node** head) {
    struct Node* newNode = (struct Node*) malloc
                           (sizeof(struct Node));
    int newVal;
    scanf("%d", &newVal);
    newNode->data = newVal;
    newNode->next = NULL;
    if (*head == NULL){
        *head = newNode;
        return;
    }
    struct Node* tail = *head;
    while (tail->next != NULL) {
```

```
        tail = tail->next;
    }
    tail->next = newNode;
}

void display(struct Node* head){
    struct Node* temp = head;
    while (temp != NULL){
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL");
}

void sortList(struct Node** head){
    struct Node* current = *head;
    struct Node* temp = NULL;
    while (current != NULL){
        temp = current->next;
        while (temp != NULL){
            if (current->data > temp->data){
                int swap = current->data;
                current->data = temp->data;
                temp->data = swap;
            }
            temp = temp->next;
        }
        current = current->next;
    }
}
```

```
void reverseList (struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current -> next;
        current -> next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

void concatenate (struct Node* head1, struct Node* head2) {
    struct Node* temp = head1;
    while (temp -> next != NULL) {
        temp = temp -> next;
    }
    temp -> next = head2;
}

int main () {
    int l1, l2;
    printf ("Enter length of First linked list:");
    scanf ("%d", &l1);
    printf ("Enter first linked list:");
    for (int i=0; i<l1; i++)
    {
        insertAtEnd (&head1);
    }
```

```
printf("First linked list : ");
display(head1);
printf("\n Second linked list :");
display(head2);
sortList(&head1);
printf("\n Sorted linked list1: ");
display(head1);
reverseList(&head2);
printf("\n Reversed linked list2 : ");
display(head2)
concatenate(head1, head2);
printf("\n Concatenated linked list : ");
display(head1);
return 0;
}
```

Output:

```
Enter length of first linked list : 5
Enter first linked list:
5 4 3 2 1
Enter second linked list : 5
Enter second linked list:
1 4 2 3 5
First linked list: 5 → 4 → 3 → 2 → 1 → NULL
Second linked list: 1 → 4 → 2 → 3 → 5 → NULL
Sorted linked list: 1 → 2 → 3 → 4 → 5 → NULL
Reversed linked list: 5 → 3 → 2 → 4 → 1 → NULL
Concated linked list: 1 → 2 → 3 → 4 → 5 → 5 → 3 → 2 → 4 → 1
→NULL
```

Q WAP to implement Singly linked list to stimulate Stack & Queue implementation

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

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

struct Node* top = NULL;

void push(int element) {
  struct Node* newNode = (struct Node*) malloc
                         (sizeof(struct Node));
  newNode -> data = element;
  newNode -> next = top;
  top = newNode;
}

void pop() {
  if (top == NULL) {
    printf("stack is empty \n");
    return;
  }
  struct Node* temp = top;
  top = top -> next;
  free(temp);
}
```

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

int main() {
    int ch;
    while (1) {
        printf("1. Push \n");
        printf("2. Pop \n");
        printf("3. Display \n");
        printf("4. Exit \n");
        printf("Enter choice");
        scanf("%d", &ch);
        switch (ch) {
            case 1: { int ele;
                      printf("Enter element: ");
                      scanf("%d", &ele);
                      push(ele);
                      break;
            case 2: pop();
                    break;
```

```
        case 3: display();
                break;
        default: return 0;
    }
}
  return 0;
}
```

Output:

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter element: 42
Enter choice: 1
Enter element: 44
Enter choice: 3
Stack: 44  42
Enter choice: 2
Enter choice: 3
Stack: 42
Enter choice: 4
Program exited
```

Queue:

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

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

struct Node* front = NULL;
struct Node* rear = NULL;

void enqueue(int element){
    struct Node* newNode = (struct Node*)malloc
                           (sizeof(struct Node));
    newNode->data = element;
    newNode->next = NULL;
    if(rear == NULL){
        front = rear = newNode;
        return;
    }
    rear->next = newNode;
    rear = newNode;
}

void dequeue(){
    if(front == NULL){
        printf("Queue is empty\n");
        return;
    }
```

```
    struct Node* temp = front;
    front = front -> next;

    if (front == NULL)
        rear = NULL;
    free(temp);
}

~~int getfront~~
void display(){
    if (front == NULL){
        printf("Queue is empty \n");
        return;
    }
    struct Node* temp = front;
    printf("Queue : \n");
    while(temp != NULL){
        printf("%d ", temp -> data);
        temp = temp -> next;
    }
    printf("\n");
}

int main(){
    int ch;
    while(1){
        printf("1. Enqueue \n");
        printf("2. Dequeue \n");
        printf("3. Display \n");
        printf("4. Exit \n");
        printf("Enter choice ");
```

```
        scanf("%d", &ch);
        switch(ch){
            case 1: { int ele;
                      printf("Enter element: ");
                      scanf("%d", &ele);
                      enqueue(ele);
                      break; }
            case 2: dequeue();
                    break;
            case 3: display();
                    break;
            default: return 0;
        }
    }
    return 0;
}
```

Output:

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter element: 33
Enter elementchoice: 1
Enter element: 54
Enter choice: 3
Queue: 33 54
Enter choice: 2
Enter choice: 3
Queue: 54
```

01-02-2024

WAP to implement a doubly linked list with primitive operations
a) Create a doubly linked list.
b) Insert a new node to the left
c) Delete a node based on value.
d) Display list

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

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

struct Node* createNode(){
    struct Node* newNode = (struct Node*) malloc (
                                sizeof (struct Node));
    int data;
    printf("Enter data in node");
    scanf("%d", &data);
    newNode -> data = data;
    newNode -> prev = NULL;
    newNode -> next = NULL;
    return newNode;
}
```

```
void insertNode (struct Node* *head){
    int pos;
    printf("Enter position of new node");
    scanf("%d", &pos);
    struct Node* newNode = createNode();
    struct Node* temp = (*head);
    while (--pos){
        if (temp->next != NULL)
            temp = temp->next;
        else {
            printf("List too short");
            return;
        }
    }
    if (temp->next == NULL){
        newNode->next = *head;
        (*head)->prev = newNode;
        (*head) = newNode;
    } else {
        temp->prev->next = newNode;
        newNode->prev = temp->prev;
        temp->prev = newNode;
        newNode->next = temp;
    }
}

void deleteNode (struct Node* *head){
    int data;
    printf("Enter data in node to be deleted");
    scanf("%d", &data);
```

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

void display(struct Node* head){
    struct Node* current = head;
    printf("Doubly linked list: ");
    while(current != NULL){
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
```

```
int main() {
    int choice;
    struct Node* head = NULL;
    while (1) {
        printf("1. Create a list \n");
        printf("2. Insert a node \n");
        printf("3. Delete a node \n");
        printf("4. Display \n");
        printf("5. Exit \n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: head = createNode();
                    break;
            case 2: insertNode(&head);
                    break;
            case 3: deleteNode(&head);
                    break;
            case 4: display(head);
                    break;
            default: printf("Exiting the program");
                     return 0;
        }
    }
}
```

Output:

1. Create a list
2. Insert a node
3. Delete a node
4. Display
5. Exit

Enter choice: 1
Enter data in node: 2

Enter choice: 2
Enter position of new node: 1
Enter data in node: 4

Enter choice: 4
Doubly Linked List: 4 → 2 → NULL

Enter choice: 3
Enter data in node to be deleted: 2

Enter choice: 4
Doubly Linked List: 4 → NULL

Enter choice: 5
Program Exited

11/2/24

15-09-2024

Write a program to construct a binary search tree, traverse the tree using inorder, preorder & postorder methods, and display all elements in the tree

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

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

Node* root = NULL;

void insert (int a);
void inorder ();
void preorder ();
void postorder ();

void main(){
   int n;
   printf(" Enter n: ");
   scanf("%d", &n);
   int a[n];
   printf("Enter array: ");
   for(int i=0; i<n; i++){
       scanf("%d", &a[i]);
       insert(a[i]);
   }
```

```
printf("Preorder: ");
preorder(root);
printf("\n Inorder: ");
inorder(root);
printf("\n Postorder: ");
postorder(root);
}

void insert(int a){
Node* newNode = (Node*)malloc(sizeof(Node));
newNode->left = NULL;
newNode->right = NULL;
newNode->data = a;
Node* temp = root;
Node* pare = NULL;
while(temp != NULL){
    pare = temp;
    if(a > temp->data){
        temp = temp->right;
    }
    else if(a < temp->data){
        temp = temp->left;
    }
    else{
        printf("Invalid data");
        return;
    }
}
if(pare == NULL){
    root = newNode;
}
```

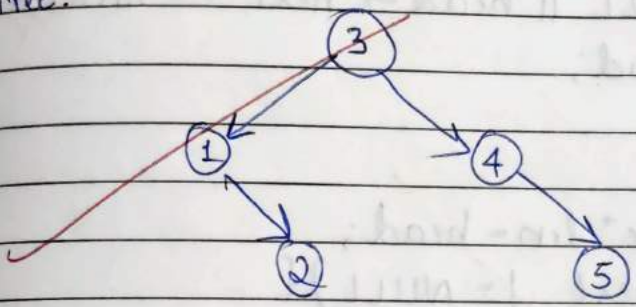
```
    else if (a < pare->data){
        pare->left = newNode;
    }
    else{
        pare->right = newNode;
    }
}

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

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

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

O/P:

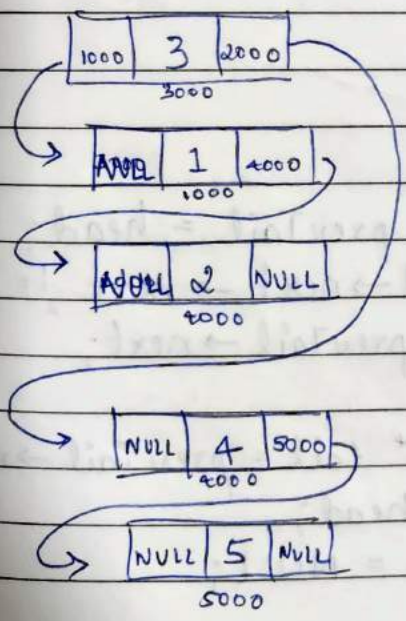
Enter n: 5
Enter array: 3 1 4 2 5

Preorder: 3 1 2 4 5
Inorder: 1 2 3 4 5
Postorder: 2 1 5 4 3

Tree:




Preorder



LEETCODE

## Rotate List

Given the head of a linked list, rotate the list to the right by k places.

```
struct ListNode* rotateRight(struct ListNode* head, int k){
    if (head == NULL || head->next == NULL){
        return head;
    }
    int n = 0;
    struct ListNode* len = head;
    while (len->next != NULL){
        n++;
        len = len->next;
    }
    n++;
    k = k%n;
    while (k--){
        struct ListNode* prevTail = head;
        while (prevTail->next->next != NULL){
            prevTail = prevTail->next;
        }
        struct ListNode* tail = prevTail->next;
        tail->next = head;
        prevTail->next = NULL;
        head = tail;
    }
    return head;
}
```

O/P:

head = [1,2,3,4,5]
k = 2
Output: [4,5,1,2,3]
Expected: [4,5,1,2,3]

8th
22/2/24.

22-02-2024

Q1. Write a program to traverse a graph using BFS method

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

int arr[20][20];
int visited[20];
int queue[20];
int front = -1;
int rear = -1;

void bfs(int start, int n){
    visited[start] = 1;
    queue[++rear] = start;

    while(front != rear){
        int current = queue[++front];
        printf("%d ", current);

        for(int i=0; i<n; i++){
            if(arr[current][i] == 1 && visited[i]){
                visited[i] = 1;
                queue[++rear] = i;
            }
        }
    }
}
```

```
void main(){
  int n, m;
  printf("Enter the number of nodes and edges: ");
  scanf("%d %d", &n, &m);
  printf("Enter the number of nodes and edges: ");
  scanf("%d %d", ,
  printf("Enter the edges: \n");
  for(int i=0; i<m; i++){
     int a, b;
     scanf("%d %d", &a, &b);
     arr[a][b] = 1;
     arr[b][a] = 1;
  }
  int start;
  printf("Enter the starting node: ");
  scanf("%d", &start);
  printf("BFS traversal: ");
  bfs(start, n);
}
```

O/P:

Enter the number of nodes and edges: 5 5
Enter the edges: 0 1 1 2 2 3 3 4 4 0
Enter the starting node: 2
BFS Traversal: 2 1 3 0 4

Q2. Write a program to check whether given graph is connected or not using DFS method.

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

int arr[20][20];
int visited[20];

void dfs(int start, int n){
    visited[start] = 1;
    for(int i=0; i<n; i++){
        if(arr[start][i] == 1 && !visited[i]){
            dfs(i, n);
        }
    }
}

int isConnected(int n){
    dfs(0, n);
    for(int i=0; i<n; i++){
        if(!visited[i]){
            return 0;
        }
    }
    return 1;
}

int main(){
    int n, m;
    printf("Enter the number of nodes and edges: ");
```

```
scanf("%d %d", &n, &m);
printf("Enter the edges: \n");
for (int i = 0; i < m; i++){
    int a, b;
    scanf("%d %d", &a, &b);
    arr[a][b] = 1;
    arr[b][a] = 1;
}
if (isConnected(n)){
    printf("The graph is connected. \n");
} else {
    printf("The graph is not connected. \n");
}
return 0;
}
```

O/P :

Enter the number of nodes and edges : 5 4
Enter the edges: 1 2 2 4 2 3 3 4
The graph is not connected.

22/2/24

29/02/24

Q Given a file of N employee records with a set K of keys which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L as integers. Design & develop a program in C that uses Hash function H: K → L as H(K) = K mod m and implement hashing technique to map a given key K to the address space L.

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

int hash_table[TABLE_SIZE] = {0};

void insert (int key) {
    int i=0;
    int hkey = key % TABLE_SIZE;
    int index = hkey;
    while (hash_table[index] != 0){
        i++;
        index = (hkey + i) % TABLE_SIZE;
        if (i == TABLE_SIZE){
            printf ("Hash table is full!\n");
            return;
        }
    }
    hash_table[index] = key;
}
```

```
void search (int key){
  int i = 0;
  int hkey = key % TABLE_SIZE;
  int index = hkey;
  while (hash_table [index] != key){
     i++;
     index = (hkey + i) % TABLE_SIZE;
     if (i == TABLE_SIZE){
        printf ("Element not found in hashtable!\n");
        return;
     }
  }
  printf ("Element found at index %d \n", index);
}

int main (){
  int choice;
  int key;
  while (1){
     printf ("1. Insert \n");
     printf ("2. Search \n");
     printf ("3. Exit \n");
     printf ("Enter choice: ");
     scanf ("%d", &choice);
     switch (choice){
        case 1: printf ("Enter Key: ");
                scanf ("%d", &key);
                insert (key);
                break;
        case 2: printf ("Enter Key: ");
                scanf ("%d", &key);
```

```
            search(key);
            break;
        default: return 0;
    }
  }
  return 0;
}
```

Output:

```
1. Insert
2. Search
3. Exit
Enter choice: 1
Enter key: 25

Enter choice: 1
Enter key: 15

Enter choice: 2
Enter key: 25
Element found at index 5

Enter choice 2
Enter key: 15
Element found at index 6

Enter choice: 3
```

Hackerrank

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

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

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

void inOrderTraversal (Node* root, int *result, int* index) {
    if (root == NULL) return;
    inOrderTraversal (root -> left, result, index);
    result[(*index)++] = root -> data;
    inOrderTraversal (root -> right, result, index);
}
```

Output:

3

2 3

-1 -1 -1

2

⇒ 3 1 2

2 1 3