Prerana

3D

1BH22CS209

DBMS LAB 1

# INDEX

NAME : Prerana NK STD : SEC : 3D ROLL NO :

| S. No. | Date | Title | Page No. | Teacher's Sign / Remarks |
|---|---|---|---|---|
| 1 | 21/12/23 | Week 1 lab 1 (i) Swapping of 2 no. (ii) Dynamic Memory Alloc (iii) Stack Implementations using Array | | |
| 2 | 28/12/23 | Week 2 lab 2 | | |
| | | (i) Infix to Postfix | | |
| | | (ii) Postfix Evaluation | | |
| 3 | 11/1/24 | Week 3 (i) Queue Implementation using Array | | |
| | | (ii) Circular Queue | | |
| | | (iii) Linked list Insertion | | |
| 4 | 3/1/24 | Week 4 (i) linked list deletion & display functions | | |
| 5 | 13/1/24 | Week 5 (i) Concatenation / reversal and searching of linked list | | |
| 6 | 25/1/24 | Week 6 (i) Stack Implementation (ii) Queue Implementation using linked list | | |
| 7 | 8/2/24 | Week 7 : (i) Implementation of Doubly linked | | |

# INDEX

Name _____ Std _____ Sec _____

Roll No. _____ Subject _____ School/College _____

School/College Tel. No. _____ Parents Tel. No. _____

Prerana
3D
1BM22CS209
DBMS LAB2

| Sl. No. | Date | Title | Page No. | Teacher Sign / Remarks |
|---|---|---|---|---|
| 8 | 15/1/24 | Week 8: Binary Search Tree (Inorder, PreOrder, Post order) | | |
| 9 | 22/1/24 | Week 9: i) Breadth first search (BFS) ii) Depth first Search (DFS) | | Sp.f |
| 10 | 29/1/24 | Week 10: i) Linear Probing | | |

int main()

21/12/2023

Week 1

① Swapping of Numbers Using Pointers

```
#include <stdio.h>
void swap (int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    int num1, num2;
    printf("Enter first and second number: ");
    scanf("%d%d", &num1, &num2);
    printf("Before swapping:")
    printf("%d\t%d\n", num1, num2);

    swap(&num1, &num2)
    printf("After swapping:")
    printf("%d\t%d\n", num1, num2);

    return 0;
}
```

→ output

Enter the first and second number : 9 2

Before swapping : 9 2

After swapping : 2 9

(2) Write a program to facilitate dynamic memory allocation [malloc, calloc, free, realloc]

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int main()
{
    int *ptr;
    int n, i;

    n=5;
    printf("Enter 5 elements : %d \n");
    ptr= (int *) calloc (n, sizeof (int));

    if (ptr == NULL)
    {
        printf ("Memory not allocated");
        exit(0);
    }
    else
    {
    printf ("Memory succesfully allocated using
            calloc");

    for (i=0 ; i<n; ++i) {
        printf ("%d", ptr[i]);
        ptr [i]= i+1;
    }
    printf ("The elements of the array are: ");
    for(i=0; i<n ; ++i)
    {
```

int main()

```
        printf("%d", ptr[i]);
    }
    n = 10;
    printf("Enter the size of the array : %d\n", n);
    ptr = (int *) realloc (ptr, n * sizeof(int));
    printf("Memory successfully re-allocated\n");
    for (i = 0; i < n; ++i){
        ptr[i] = i + 1;
    }
    printf("The elements of the array are: ");
    for (i=0; i<n; ++i) {
        printf("%d", ptr[i]);
    }
    free(ptr);
}
return 0;
}
```

→ Output

Enter number of elements : 5
Memory successfully allocated using calloc
The elements of the array 1, 2, 3, 4, 5
Enter the new size of array : 10
Memory successfully reallocated using realloc

(5) Stack Implementation using Arrays

```
#include <stdio.h>
#define SIZE 5
int stack[SIZE];
int top = -1;
void push (int element);
void pop();
void display ();
int main () {
    int choice, element;
    do {
        printf ("Stack Operations \n");
        printf ("1. Push \n");
        printf ("2. Pop \n");
        printf ("3. Display \n");
        printf ("4. Exit \n");
        printf ("Enter your choice");
        scanf ("%d", &choice);
        switch choice {
        case 1: printf ("Enter elements to push");
                scanf ("%d", &element);
                push (element);
                break;
```

```
int main()

        case 2 : pop()
                 break;

        case 3 :  display()
                  break;

        case 4: printf("Exiting the program.\n");
                break;

        default :
                printf("Invalid choice");

    }

} while (choice != 4);

return 0;

}

void push (int element) {
    if (top == SIZE - 1) {
            printf ("Stack overflow! Cannot push
                     element.\n");

    }

    else {
        top++;
        stack[top] = element;
        printf("%d pushed onto stack\n",
                element);
    }

}
```

```
void pop() {
    if (top == -1)
    {
        printf("Stack Underflow! Cannot pop
                elements\n");
    }
    else
    {
        printf("%d popped from stack\n",
                stack[top]);
        top--;
    }
}

void display() {
    if (top == -1) {
        printf("Stack is empty");
    }
    else {
        printf("Elements in the stack");
        for (int i=0; i<=top; i++) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}
```

int main ()

→ Output

→ Stack Operations

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

enter your choice : 1

enter element to push : 5

5 pushed onto the stack

→ Stack Operations

1 Push

2 Pop

3 Display

4 Exit

21/12/2023

enter your choice : 1

enter the element to push : 9

9 pushed onto the stack

→ Stack Operations :

1 Push

2 Pop

3 Pull

4 Exit

enter your choice : 2

9 popped from the stack

→ Stack Operations

1 Push

2 Pop

3 Display

4 Exit

Enter your choice: 3

Elements in stack : 5

→ Stack Operations

1 Push

2 Pop

3 Display

4 Exit

Enter your choice: 4

Exiting the program

int main ()

28/12/23 Week 8

(2) Infix to Postfix

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

#define SIZE 50

char stack [SIZE];
int top = -1;

void push (char element)
{
        stack [++top] = elem;
}

char pop ()
{
        return (stack [top--]);
}

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

```
int main()
{
    char infix[50], postfix[50], ch, elem;
    int i=0, k=0;
    printf("Enter the infix expression:");
    scanf("%s", infix);

    push('#');

    while((ch = infix[i++]) != '\0')
    {
        if(ch == '(')
            push ch;
        else if(isalnum(ch))
            postfix[k++] = ch;
        else if(ch == ')')
        {
            while(stack[top] != '(')
                postfix[k++] = pop();

            elem = pop();
        }
        else
        {
            while(top != -1 && pr(stack[top])
                >= pr(ch))
                postfix[k++] = pop();
            push(ch);
        }
    }
```

```
int main ()

    while (stack[top] != '#')
        postfix[k++] = pop();

    postfix[k] = '\0';
    printf("\n Postfix expression : %s \n", postfix);

    return 0;

}
```

→ Output

Enter infix expression : A + B - (C/D) * (E^F)

Postfix expression : A B + C D / E F ^ * -

Fig. 1.

→ Postfix Evaluation

```
#include <stdio.h>
int stack[20];
int top = -1;

void push (int x)
{
    stack[++top] = x
}
int pop()
{
    return stack[top--];
}
int main()
{
    char exp[20];
    char *e;
    int n1, n2, n3, num;
    printf("Enter the expression");
    scanf("%s", exp);
    e = exp;
    while(*e != '\0')
    {
        if(isdigit(*e))
        {
            num = *e - 48;
            push(num);
        }
        else
        {
            n2 = pop();
            n1 = pop();
            switch(*e)
            {
                case '+':
                {
                    n3 = n1 + n2;
                    break;
                }
                case '-':
                {
                    n3 = n1 - n2;
                    break;
                }
                case '*':
                {
                    n3 = n1 * n2;
                    break;
                }
                case '/':
                {
                    n3 = n1 / n2;
                    break;
                }
            }
            push(n3);
        }
        e++;
    }
    printf("\n The result of exp %s = %d\n\n", exp, pop());
}
```

→

output

231*+9-

The result of expression

231*+9- = -4

① 17/07/2020 Linked list (functions) work3

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

struct node
{
    int data;
    struct node * next;
};
struct node * create node(int value)
struct node *new = (struct node *) malloc (sizeof (struct node));
if (newnode == NULL)
{
    printf("Memory allocation failed \n");
    exit(1);
}
newnode->data = value;
newnode->next = NULL;
return newnode;
}
struct node * insertAtBeginning(struct node * head, int value)
{
  struct node * newnode = create node(value);
  newnode->next = head;
  return newnode;
}
struct node * insertAtPos(struct node * head, int value, int pos)
{
   struct node * newnode = create node(value)
   struct node * temp = head;
   if (pos==1)
   {
      newnode->next=head;
      return newnode;
```

(2)

```
int i;
for(i=1; i<pos-1; i++)
    temp = temp -> next;
}
newNode -> next = temp -> next;
temp -> next = newNode;
return head;
}
struct Node * insertAtEnd (structNode * head, int value)
{
    structNode * newNode = createNode (value);
    if (head == NULL)
    {
        return newNode;
    }
    struct Node * temp = head;
    while (temp -> next != NULL)
    {
        temp = temp -> next
    }
    temp -> next = newNode;
    return head;
}
void displayList (structNode * head)
{
    structNode * temp = head;
    while (temp != NULL)
    {
        printf ("%d", temp -> data);
        temp = temp -> next;
    }
    printf ("NULL\n");
}
```

```
int main()
struct node * head = NULL;
int choice, value, pos;
while(1)
{
    printf("1 Insert at end \n");
    printf("2 Insert at beginning \n");
    printf("3 Insert at any position \n");
    printf("4 Display list \n");
    printf("Enter your choice: ");
    printf("%d", &choice);
    switch (choice)
    {
    case 1: printf("enter new value to be inserted")
            scanf("%d", &value);
            head = insertAtEnd(head, value);
            break;
    case 2: printf("enter the value to be inserted");
            scanf("%d", &value);
            head = insertAtBeginning(head, value);
            break;
    case 3: printf("Enter the value to be inserted")
            scanf("%d", &value);
            printf("Enter the position");
            scanf("%d", &pos);
            head = insertAtAnyPos(head, value, pos)
            break;
    case 4: displayList(head);
            break;
```

```
case 5 : exit(0)
default : printf("Invalid choice");
}
}
return 0;
}
```

→ Output

1) Insert at end
2) Insert at beginning
3) Insert at any position
4) Display
5) Exit

Enter your choice 2
Enter the value to be inserted : 12

Enter your choice : 2
Enter the value to be inserted : 13

Enter your choice : 4
13 → 12 → NULL

Enter your choice : 1
Enter the value to be inserted : 23
Enter your choice : 4
13 → 12 → 23 → NULL

Enter your choice : 3
Enter the value to be inserted : 12
Enter the position to insert : 3
Enter your choice : 4
13 → 12 → 12 → 23 → NULL

~~23/07/24~~

Week 3

## Circular Queue

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

int queue[5];
int front = -1;
int rear = -1;

void enqueue (int x)
{
    if ((rear + 1) % 5 == front)
        printf ("Queue is full");

    else if (front == -1 && rear == -1)
    {
        front = rear = 0;
        queue[rear] = x
    }
    else
    {
        rear = (rear + 1) % 5;
        queue[rear] = x;
    }
}

void dequeue ()
{
    if (front == -1 && rear == -1)
        printf ("Queue is empty")
    else if (front == rear)
        front = rear = -1;
    else
```

```
    }
        printf("%d", queue[front]);
        front = ((front + 1) % N);
    }
}

void display()
{
    if (front == -1 && rear == -1)
        printf("Queue is empty");
    else if (front == rear)
    {
        int i;
        while (i != rear)
        {
            printf("%d", queue[i]);
            i = (i+1) % N;
        }
        printf("%d", queue[rear]);
    }
}

void peek()
{
    printf("%d", queue[front]);
}

void main()
{
    int ch;
    int x;
```

```
do
{
    printf("Enter your choice");
    scanf("%d", &choice);
    printf("Press 1: Enqueue  2: Dequeue  3: Peek  4: Display");
    switch(ch)
    {
        case 1: printf("Enter x");
                scanf("%d", &x);
                enqueue(x)
                break;

        case 2: dequeue()
                break;

        case 3: peek()
                break;

        case 4: display()
                break;

        default: printf("Invalid choice");
    }
} while(ch!=0)
}
```

→ output

Press 1: Enqueue 2: Dequeue 3: Peek 4: Display

Enter your choice: 1

Enter the x: 23

Press 1: Enqueue 2: Dequeue 3: Peek 4: Display

Enter your choice: 1

Enter x: 45

Press 1: Enqueue 2: Dequeue 3: Peek 4: Display

Enter your choice: 1

Enter x: 28

Press 1: Enqueue 2: Dequeue 3: Peek 4: Display

Enter your choice: 2

~~Exit~~

Press 1: Enqueue 2: Dequeue 3: Peek 4: Display

Enter your choice: 4

45, 28

Sp.1
18/1/24

Deletion linked list Week 4

11/10/23

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

struct node
{
    int data;
    struct node * next;
};
void create_ll (struct node ** start);
void display (struct node *start);
void top (struct node **start);
void end_delete (struct node **start)
void delete_at_pos (struct node ** start);
void full_list (struct node * start);

int main (void)
{
    struct node * start = NULL;
    int option;

    do
    {
        printf ("\n\n *** MAIN MENU ***");
        printf ("\n 1 Create a list");
        printf ("\n 2 Display the list")
        printf ("\n 3 Delete a node from the
                beginning");

        printf ("\n4 Delete a node from the
                end");

        printf ("\n 5 Delete from a specific position");
```

```
printf("\n 6 Exit");
printf("\n Enter your option");
scanf("%d", &option);

switch(option)
{
    case 1: create_ll(&start);
            printf("\n Linked list created");
            break;

    case 2: display(start);
            break;

    case 3: pp(&start);
            break;

    case 4: end_delete(&start);
            break;

    case 5: delete_at_pos(&start)
            break;

    case 6: free_list(start);
            printf("\n Exiting...\n");
            break;

}
}while(option != 6)

return 0;
}
```

```
void create_ll (struct node ** start)
{
    struct node * new_node, * ptr;
    int num;
    printf("Enter -1 to end \n");
    printf( "Enter the data");
    scanf ("%d", &num);

    while (num != -1)
    {
        new_node = (struct node *) malloc (sizeof(struct node));
        if (new_node == NULL)
        {
            printf( "Memory allocation failed \n");
            exit (EXIT_FAILURE);
        }

        new_node -> data = num;
        new_node -> next = NULL;

        if ( * start == NULL)
        {
            * start = new_node;
        }
        else
        {
            ptr = * start;
            while (ptr -> next != NULL)
                ptr = ptr -> next;
            ptr -> next = new_node;
        }
```

```
printf ("\n Enter the data : ");
scanf ("%d", &num);
}
}
void display (struct node *start)
{
    struct node *ptr = start;
    while (ptr != NULL)
    {
        printf ("\t %d", ptr -> data);
        ptr = ptr -> next;
    }
}
void pop (struct node **start)
{
    if (*start == NULL)
    {
        printf ("List is empty\n");
        return;
    }
    struct node *ptr = *start;
    *start = (*start) -> next;
    free (ptr)
}
```

```
void end_delete (struct node **a start)
{
    if (*start == NULL)
    {
        printf ("list is empty");
        return;
    }
    struct node *ptr = a start
    struct node *ptr1 = NULL
    while (ptr -> next != NULL)
    {
        ptr1 = ptr;
        ptr = ptr -> next;
    }
    if (ptr1 != NULL)
    {
        ptr1 -> next = NULL;
        free (ptr)
    }
    else
    {
        free (ptr)
        *start = NULL;
    }
}

void delete_at_pos (struct node **a start)
{
    if (*start == NULL)
    {
        printf ("list is empty\n");
        return;
    }
```

```
int loc
printf("\n Enter the location of the node which has to
        be deleted");
scanf("%d", &loc);
struct node *ptr = &start
struct node *ptr1 = NULL

for (int i = 0; i < loc; i++)
{
    ptr1 = ptr
    ptr = ptr -> next;
    if (ptr == NULL)
    {
        printf("There are less than %d elements in list\n", loc);
        return;
    }
}
if (ptr1 != NULL)
{
    ptr -> next = ptr -> next
    free(ptr)
    printf("Deleted node at %d position\n", loc);
}
else
{
    start = ptr -> next;
    free(ptr);
    printf("Delete node at %d position\n", loc);
}
```

```
void free_list (struct node * start)
{
    struct node * ptr = start;
    struct node * next_node;

    while (ptr != NULL)
    {
        next_node = ptr -> next;
        free(ptr);
        ptr = next_node;
    }
}
```

→ output

Main Menu

1. Create a list
2. Display the list
3. Delete node from beginning
4. Delete node from end
5. Delete from specific position
6. Exit

Enter your option : 1

Enter -1 to end

Enter data : 10

Enter data : 20

Enter data : 30

Enter data : 40

Enter data : -1

Enter your option : 2

10 20 30 40

Enter your option : 3

Enter your option:

20 30 40

Enter your option : 4

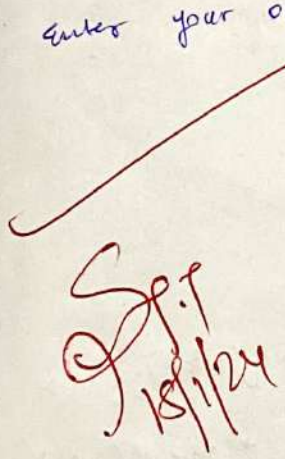
20 30

Enter your option : 5

Enter the location of node to be deleted : 1

Enter your option : 2

20



LEET Code  Min Stack

```
typedef struct {
    int size;
    int top;
    int *s;
    int * minstack;
} Minstack;

Minstack * minStackCreate() {
    Minstack * st = (Minstack*) malloc (sizeof (MinStack));
    if (st== NULL)
    {
        printf ("memory allocation failed");
        exit(0);
    }
    st -> size= 5;
    st -> top=-1;
    st -> s = (int *) malloc (st -> size* sizeof (int));
    st -> minstack= (int *) malloc (st -> size * sizeof (int));
    if (st -> s == NULL)
    {
        printf ("memory allocation failed");
        free (st -> s);
        free (st -> minstack);
        exit(0);
    }
    return st;
}
```

```
void minStackPush (MinStack * obj, int val)
{
    if (obj -> top == obj -> size-1)
    {
        printf ("stack is overflow");
    }
    else
    {
        obj -> top ++;
        obj -> s [obj -> top] = val;
        if (obj -> top == 0 || val < obj -> minstack
                                    [obj -> top - 1])
        {
            obj -> minstack [obj -> top] = val;
        }
        else
        {
            obj -> minstack [obj -> top] = obj -> minstack
                                    [obj -> top - 1];
        }
    }
}

void minStackPop (MinStack * obj)
{
    int value = -1;
    if (obj -> top == -1)
    {
        printf ("underflow \n");
        exit(0);
    }
```

```
    else
    {
        value = obj -> s [obj -> top];
        return value;
    }
}

int minStackGet Min (MinStack * obj)
{
    if (obj -> top == -1)
    {
        printf ("underflow \n");
        exit(0);
    }
    else
    {
        return obj -> minstack [obj -> top];
    }
}
void minstackFree (MinStack * obj)
{
    free (obj -> s);
    free (obj -> minstack);
    free (obj);
}
```

→ output

Input

["MinStack", "push", "push", "push", "getMin",

"pop", "top", "getMin"]

stdout

-3 is popped

Output

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

Expected

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

Lut Code 2

```
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++)
{
        a = a -> next;
}
difference -= 2;
}
return head;
}
```

→ output

head = [1, 2, 3, 4, 5]

left = 2

right = 4

stdout

2

4

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

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

Week 5

① Implementation / Searching and Traversal of a linked list.

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

struct Node
{
    int data;
    struct Node * next;
};
typedef struct Node node;
node * makeNode(int value)
{
    Node * newNode = (node*) malloc (sizeof (node));
    newNode -> data = value;
    newNode -> next = NULL;
    return newNode;
}

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

```
Node * sortlist (Node * head)
{
    if (head == NULL || head -> next == NULL)
        return head;
    int swapped;
    Node * temp;
    Node * end = NULL;
    do
    {
        swapped = 0;
        temp = head;
        while (temp -> next != end)
        {
            if (temp -> data > temp -> next -> data)
            {
                int tempData = temp -> data;
                temp -> data = temp -> next -> data;
                temp -> next -> data = tempData;
                swapped = -1;
            }
            temp = temp -> next;
        }
        end = temp;
    } while (swapped);
    return head;
}
```

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

Node * concatLists ( Node * list1, Node * list2)
{
    if (list1 == list2)
        return list2;

    Node * temp = list1;
    while ( temp -> next != NULL)
    {
        temp = temp -> next;
    }
    temp -> next = list2;
    return list1;
}
```

```
void main()
{
    Node * list1 = create Node (3);
    list1 -> next = create Node (1);
    list1 -> next -> next = create Node (4);

    Node * list2 = create Node (2);
    list2 -> next = create Node (5);

    printf ("original list 1:");
    display (list1);
    printf ("original list 2:");
    display (list2);

    list1 = sort list (list1);
    printf ("sorted list 1:");
    display (list1)

    list1 = reverse list (list1);
    printf ("sorted list 1:");
    display (list1);

    Node * concatanated = concat lists (list1, list2);
    printf ("concatanated list:");
    display (concatanated);
}
```

→ output

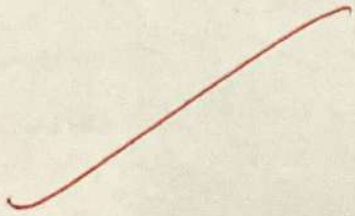
original list: 1 → 3 → 1 → 4 → NULL

original list 2: 2 → 5 → NULL

sorted list 1: 1 → 3 → 4 → NULL

reversed list 1: 4 → 3 → 1 → NULL

concatenated list: 4 → 3 → 1 → 2 → 5



→ Stack implementation using singly linked list

Week 6

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

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

struct Node * head = NULL;

void push (struct Node ** head_ref, int new_data)
{
    struct Node* new_node = (struct Node*) malloc (sizeof (struct Node));
    if (new_node == NULL)
    {
        printf ("Memory allocation failed");
        exit (1);
    }
    new_node -> data = new_data;
    new_node -> next = * head_ref;
    *head_ref = new_node;
    printf ("%d pushed to stack \n", new_data);
}

void pop()
{
    struct Node * ptr;
    if (head == NULL)
    {
        printf ("stack is empty");
    }
    else
    {
```

```
        ptr = head;
        ptr = ptr -> next;
        printf("%d popped from the stack\n", ptr -> data);
    }
}

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

int main()
{
    int choice, data;
    do
    {
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("0. Exit\n");
        printf("Enter your choice");
        scanf("%d", &choice);
```

```
switch (choice)
{
    case 1: printf("Enter data to push:");
            scanf("%d", &data);
            push(&head, data);
            break;

    case 2: pop();
            break;

    case 3: display();
            break;

    case 0: printf("Exiting program\n");
            break;

    default: printf("Invalid choice\n");

}
} while (choice != 0);

struct Node * current = head;

while (current != NULL)
{
    struct Node* next = current->next;
    free(current);
    current = next;
}

return 0;
}
```

→ output

1 Push

2 Pop

3 Display

4 Exit

Enter your choice : 1
Enter data to push: 2

Enter your choice : 1
Enter data to push: 3

Enter your choice : 1
Enter data to push: 4

Enter your choice: 3
Stack : 4 3 2

Enter your choice: 2
4 popped from stack

Enter your choice: 3
Stack : 3 2

Enter your choice : 0

Exiting program

## Queue using singly linked list

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

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

typedef struct Node Node;

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

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

typedef struct
{
    Node * front;
    Node * rear;
} LinkedList;

void enqueue (LinkedList * queue, int value)
{
```

```
    Node * newNode = createNode(value);
    if (queue -> front = NULL)
    {queue -> front = newNode;
     queue -> rear = newNode;
    }
    else
    {
        queue -> rear -> next = newNode;
        queue -> rear = newNode;
    }
}

int dequeue (linkedList * queue)
{
    if (queue -> front == NULL)
    {
        printf ("queue is empty:\n");
        return -1;
    }
    int dequeuedvalue = queue->front -> data;
    Node * temp = queue -> front;
    queue -> front = queue -> front -> rear
    free (temp);
    return dequeued value;
}

void main()
{
    linkedList queue;
    queue.front = NULL;
    queue.rear = NULL;
    enqueue (&queue, 40);
    enqueue (&queue, 60);
    enqueue (&queue, 80);
```

```
display(queue.front);
printf("dequeued from queue: %d\n", dequeue(&queue));
printf("dequed from queue: %d\n", dequeue(&queue));
display(queue.front);
}
```

→ output

queue operations:

40 → 60 → 80 → NULL

dequeue from queue: 40

dequeue from queue: 60

80 → NULL

Sat
25/2/24

## → Doubly linked list

Week 7

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node * prev;
    struct Node * next;
};
struct Node * createNode (int data)
{
struct Node * newNode = (struct Node*) malloc (sizeof (struct Node));
if (newNode == NULL)
{
    printf("memory allocation failed");
    exit (1);
}
newNode -> data = data;
newNode -> prev = NULL;
newNode -> next = NULL;
return newNode;
}

void insertNodeToLeft (struct Node* head, struct Node * target, int data)
{
    struct Node * newNode = createNode (data);
    if (target -> prev != NULL)
    {
        target -> prev -> next = newNode;
        newNode -> prev = target -> prev;
    }
    else
    {
        head = newNode;
    }
    newNode -> next = target;
    target -> prev = newNode;
}
void deleteNode (struct Node * head, int value)
{
    struct Node * current = head;
    while (current != NULL)
    {
        if (current -> prev != NULL)
        {
            current -> next -> prev = current -> prev;
        }
        free (current);
    return;
```

NL 1/2/24

```
        }
        current = current -> next;
    }
    printf ("Node with value %d not found \n");
}

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

int main()
{
    struct Node * head = NULL;
    head = create Node(1);
    head -> next = create Node (2);
    head -> next -> prev = head;
    head -> next -> next = create Node (3);
    head -> next -> next -> prev = head -> next;
    display list (head);

    insert NodeToLeft (head, head -> next, 10);
    printf ("After deletion: \n");
    display list (head);

    return 0;
}
```

→ output

Doubly linked list: 1 ⟷ 2 ⟷ 3 ⟷ NULL

After insertion

Doubly linked list: 1 ⟷ 10 ⟷ 2 ⟷ 3 ⟷ NULL

After Deletion

Doubly linked list: 1 ⟷ 10 ⟷ 3 ⟷ NULL

Prerana
3D
1BM22CS209
DBMS Lab 2

# INDEX

Name _____ Std _____ Sec _____

Roll No. _____ Subject _____ School/College _____

School/College Tel. No. _____ Parents Tel. No. _____

| Sl. No. | Date | Title | Page No. | Teacher Sign / Remarks |
|---|---|---|---|---|
| 8 | 15/1/24 | Week 8: Binary Search Tree (Inorder, PreOrder, Post Order) | | |
| 9 | 24/1/24 | Week 9: (i) Breadth First Search (ii) Depth First Search | (BFS) (DFS) | Sp.f |
| 10 | 29/1/24 | Week 10: (i) Linear Probing | | |

# Week 8

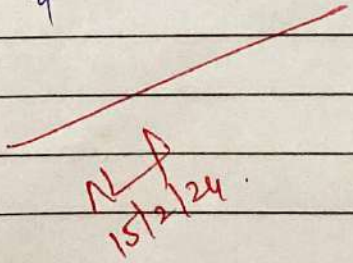
13/02/24

Write a C Program to construct a Binary Search Tree. Traverse using i) Recorder (ii) Inorder (iii) Postord

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

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

node * create() {
    node * temp;
    printf("\n Enter data: ");
    temp = (node *) malloc (sizeof (node));
    scanf ("%d", &temp -> data);
    temp -> left = temp -> right = NULL;
    return temp;
}

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

Date __/__/__
Page _____

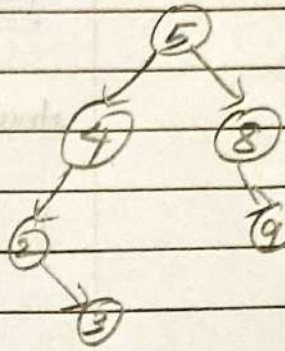
```
void preorder (node * root)
{
    if (root != NULL)
    {
        printf ("%d", root -> data);
        preorder (root -> left);
        preorder (root -> right);
    }
}

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);
    }
}
```



```
int main() {
    char ch;
    node *root = NULL, *temp;

    do {
        temp = create();
        if (root == NULL)
            root = temp;
        else
            insert(root, temp);
        printf("Do you want to enter more?");
        getchar();
        scanf("%c", &ch);
    } while (ch == 'y' || ch == 'Y');

    printf("\n Preorder Traversal:");
    preorder(root)

    printf("\n Inorder Traversal:");
    inorder(root)

    printf("\n Postorder Traversal:");
    postorder(root);

    return 0;
}
```

Date __/__/__
Page _____

→ Output




Enter data: 4

Do you want to enter more? (y/n) : y

Enter data: 2

Do you want to enter more? (y/n) : y

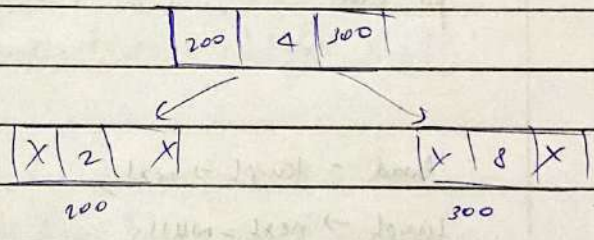
Enter data: 8

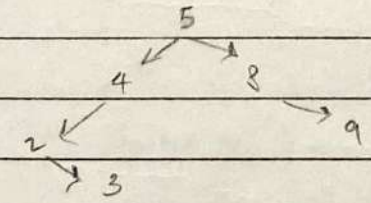
Do you want to enter data? (y/n) : n

Preorder Traversal: 4 2 8

Inorder Traversal : 2 4 8

postOrder Traversal : 2 8 4




Preorder : 5 4 2 3 8 9

Inorder : 2 3 4 5 8 9

postorder: 3 2 4 9 8 5

Date __/__/__

Page

List code

```
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;
}
```

Case 1

head =

[1, 2, 3, 4, 5]

k =

2

output

[4, 5, 1, 2, 3]

22/2/2024

Date __/__/__
Page _____

Lab Program 9

(1) Breadth First Search

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_SIZE 100

struct queue {
    int item[MAX_SIZE];
    int front;
    int rear;
};

struct Graph {
    int vertices;
    bool adjMatrix[MAX_SIZE][MAX_SIZE];
};

struct queue* createQueue() {
    struct queue* queue = (struct queue*) malloc(sizeof(struct queue));

    queue->front = -1;
    queue->rear = -1;
    return queue;
}

bool isEmpty(struct queue* queue) {
    if (queue->rear == -1)
        return true;
    else
        return false;
}
```

```
void enqueue (struct queue * queue, int value){
    if (queue -> rear == MAX_SIZE - 1);
        printf ("\n queue is full.");
    else {
        if (queue -> front == -1)
            queue -> front = 0;
        queue -> rear ++;
        queue -> items (queue -> rear) = value;
    }
}

int dequeue (struct queue * queue) {
    int item;
    if (isEmpty (queue)) {
        printf ("\n queue is empty!");
        item = -1;
    } else {
        item = queue -> items [queue -> front];
        queue -> front ++;
        if (queue -> front > queue -> rear)
        {   queue -> front = queue -> rear = -1;
        }
    }
    return item;
}
```

```
void createGraph (struct Graph * graph, int vertices ) {
    graph -> vertices = vertices;

    for (int i = 0; i < vertices ; i++ )
    {
        for (int j = 0; j < vertices ; j++)
        {
            graph -> adjMatrix [i][j] = false;
        }
    }
}

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

void BFS (struct Graph * graph, int startVertex ) {
    bool visited [MAX_SIZE] = { false };
    struct Queue * queue = createQueue();

    visited [startVertex] = true;
    enqueue (queue, startVertex);

    while (!isEmpty (queue))
    {
        int currentVertex = dequeue (queue);
        printf ("%d ", currentVertex);

        for (int i=0; i < graph -> vertices ; i++)
        {
```

Date
Page

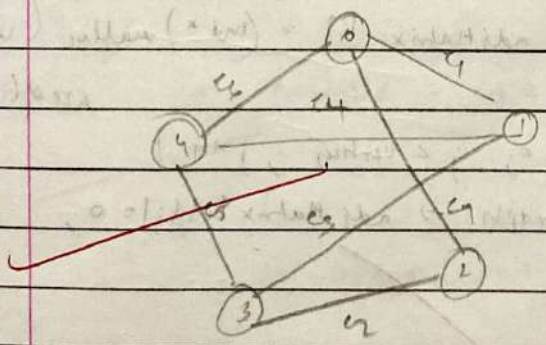
```
if (graph->adjMatrix[currentVertex][i] &&
                    !visited[i])
{
    visited[i] = true;
    enqueue(queue, i);
}
}
}
}
}

int main() {
    struct Graph* graph;
    int vertices, edges, startVertex;

    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);

    graph = createGraph(vertices);

    printf("Enter the number of edges: ");
    scanf("%d", &edges);

    for (int i = 0; i < edges; i++) {
        int src, dest;
        printf("Enter edge %d source and destination
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }
```

Date __/__/__
Page _____

```
printf("Enter the starting vertex: ");
scanf("%d", &startVertex);

printf("\nBFS Traversal ");
BFS(&graph, startVertex);

return 0;
}
```

→ OUTPUT

```
Enter the number of vertices: 5
Enter the number of edges: 7
Enter edge 1 source and destination: 0 1
Enter edge 2 source and destination: 2 3
Enter edge 3 source and destination: 1 3
Enter edge 4 source and destination: 1 4
Enter edge 5 source and destination: 3 4
Enter edge 6 source and destination: 0 4
Enter edge 7 source and destination: 0 2
```




```
Enter the starting vertex for BFS: 0

BFS traversal starting from vertex 0: 0 1 2 4 3
```

# Depth First Search

(11)

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

#define MAX_SIZE 100

struct Graph
{
    int vertices;
    int **adjMatrix;
};

struct Graph *createGraph(int vertices)
{
    struct Graph *graph = (struct Graph *) malloc
                          (sizeof(struct Graph));

    graph->vertices = vertices;

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

Date __/__/__
Page _____

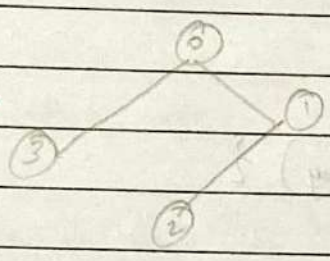
```
void addEdge (struct Graph * graph, int src, int dist)
{
    graph -> adjMatrix [src][dist] = 1;
    graph -> adjMatrix [dist][src] = 1;
}

void DFS ( struct Graph * graph, int startVertex,
                                 int visited[])
{
    visited [startVertex] = 1;

    for (int i = 0; i < graph -> vertices; i++)
    {
        if (graph -> adjMatrix [startVertex][i] == 1
            && visited [i] == 0)
            DFS (graph, i, visited);
    }
}

int isBounded ( struct Graph * graph)
{
    int * visited = (int *) malloc (graph -> vertices * sizeof (int));

    for (int i=0; i < graph -> vertices; i++)
        visited [i] = 0;

    DFS (graph, 0, visited);

    for (int i = 0; i < graph -> vertices; i++)
    {
        if (visited [i] == 0)
            return 0;
    }
}
```

```
    return 1;
}

int main()
{
    int vertices, edges, src, dest;
    printf("Enter the no. of vertices: ");
    scanf("%d", &vertices);

    struct Graph* graph = createGraph(vertices);

    printf("Enter the no. of edges: ");
    scanf("%d", &edges);

    for (int i=0; i<edges; i++)
    {
        printf("Enter edge %d (source destination): ",
               i+1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }
    if (isConnected(graph))
        printf("The graph is connected\n");
    else
        printf("The graph is connected\n");
    return 0;
}
```

Date __/__/__
Page _____

→ output

Enter the no. of vertices: 4
Enter the no. of edges: 3
Enter edge 1 (source destination) : 1 2
Enter edge 2 (source destination) : 0 3
Enter edge 3 (source destination) : 0 1
The graph is connected




Sof
22/2/24.

29/1/24 Lab Program 10

LINEAR PROBING

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

int hashTable [table_size];

void initializeHashTable() {
    for (int i=0; i < TABLESIZE ; i++ ) {
        hashTable[i] = -1;
    }
}

void insert (int key) {
    int i=0;
    int hkey= key % TABLE_SIZE;
    int index;

    do {
        index = (hkey + i) % TABLE_SIZE;
        if (hashtable [index] == -1)
        {
            (hashTable [index] == -1)
            hashTable [index] = key.
            printf ("Inserted %d at index %d\n",
                    key, index);
            return;
        }
        i++;
    } while (i < TABLE_SIZE);

    printf ("Unable to insert %d. Hash table is full. \n", key
}
```

```
int search (int key) {
    int i = 0;
    int hkey = key % TABLE_SIZE;
    int index;

    do {
        index = (hkey + i) % TABLE_SIZE;

        if (hashTable[index] == key) {
            printf("Element %d found at index
                %d\n", key, index);
            return index;
        }
        i++;
    } while (i < TABLE_SIZE);

    printf("Element %d not found in the hash table\n",
                key);
    return -1;
}

int main()
{
    initialiseHashTable();
    insert(5);
    insert(15);
    insert(25);

    search(15);
    search(20);
}
```

→ output

Inserted 5 at index 5
Inserted 15 at index 6
Inserted 25 at index 7

Element 15 found at index 6
Element 10 not found in the hash table