USN: 1BM22CS207

(WEEK-1 to WEEK 7)

INDEX

Name: Preethi - Narasimhan Std: Sec:

Roll No: Subject: DS Observation Book - 1 School/College:

School/College Tel. No: Parents Tel. No:

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1.	21/12/23	[Stack Implementation] Week 1 Lab (Swapping Dynamic)		
2.	28/12/23	Week 2 Lab a. Infix to Postfix b. Postfix Evaluation		
3.	11/1/24	Week 3 Lab a. Queue implementation with arrays		
4.		b. Circular Queue implement (with arrays) c. Linked list Insertion		
4.	18/1/24	Week-4 Lab a. Linked list deletion & display		
5.	18/1/24	Week-5 Lab a. Min Stack Leetcode pgm b. Reversing a Linked List		
6.	25/1/24	Week-6 Lab a. Sorting a singly LL b. Reverse a singly LL c. Concatenate with another LL d. Implement Stacks with LL e. Implement Queues with LL		
7.	1/2/24	Week-7 Lab a. Doubly Linked List Implementation		
8.	15/2/24	Week-8 Lab a. Rotate LL to right b. Construct binary tree & traversal		

100

10 (Main Lab Starts)

Date / / Page

WEEK 1 LAB

1. Write a program to swap 2 numbers using pointers

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
void main()
{
    int a,b;
    printf("Enter two numbers: \n");
    scanf("%d%d", &a, &b);
    printf("Numbers before swapping: \n");
    printf("%d%d", a, b);
    swap(&a, &b);
    printf("\n Numbers after swapping: \n");
    printf("%d%d", a, b);
}
```

OUTPUT:

```
enter two numbers:
3
4
Numbers before swapping:
4 3
```

Date __/__/__
Page _____

3 4
Numbers after swapping :
4 3

2. Write a program to facilitate dynamic memory allocation [malloc, ~~free~~, calloc, realloc] fns

Showing usage of 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 \n");
        exit(0);
    }
    else
    {
        printf("Memory successfully
        allocated using calloc. \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]);
        }
        n=10;
        printf("\n\n Enter the new size of
        the array : %d \n", n);
        ptr = (int *)realloc (ptr, n * sizeof(int));
        printf(" Memory successfully
        reallocated using realloc.\n");
        for(i=5; i<n; ++i)
        {
            ptr[i] = i+1;
        }
        printf(" The elements of the array are : \n");
        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 are 1, 2, 3, 4, 5.
Enter the new size of the array : 10
Memory successfully re-allocated using realloc

Date __/__/__
Page _____

Stack Implementation using Arrays

1. The elements of the array are : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

3. Write a pgm for Stack implementation using Arrays in C

```
#include <stdio.h>
#include <conio.h>
int));
#include <stdlib.h>
int stack[100], choice, n, top, x, i;
void push(void);
void pop(void);
void display(void);
int main()
{
ure:\n");
    top = -1;
    printf("\n Enter the size of
        STACK[MAX=100]: ");
    scanf("%d", &n);
    printf("\n\t STACK OPERATIONS
        USING ARRAY");
    printf("\n\t ----------.");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t
        3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the choice:");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
            {
```

```
            push()
            break;
        }
        case 2:
        {
            pop();
            break;
        }
        case 3:
        {
            display();
            break;
        }
        case 4:
        {
            printf("\n\t EXIT POINT
            break;
        }
        default:
        {
            printf("\n\t Please
            Enter a Valid choice
            (1/2/3/4)");
        }
    }
}
while(choice != 4);
return 0;
}
void pop()
{
    if(top <= -1)
```

```
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is
            %d", stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK\n");
        for(i=top; i>=0; i--)
            printf("\n%d", stack[i]);
        printf("\n Press Next Choice");
    }
    else
    {
        printf("\n The STACK is empty");
    }
}
```

OUTPUT:

Enter the size of STACK [MAX=100] : 5

STACK OPERATIONS USING ARRAY

- - - - - - - - - -

1. PUSH
2. POP
3. DISPLAY
4. EXIT

Enter the choice : 1
Enter a value to be pushed : 12

Enter the choice : 1
Enter a value to be pushed : 24

Enter the choice : 1
Enter a value to be pushed : 36

Enter the choice : 3
The elements in STACK
36
24
12
Press Next choice
Enter the choice : 2
The popped element is 36
Enter the choice : 3
The elements in STACK
24
12
Press Next choice
Enter the choice : 4
EXIT POINT

Sept
21/12/23

Date __/__/__
Page _____

Pgm 3 contnd :

```
void push()
{
    if (top >= n-1)
    {
        printf("\n\t STACK is over
                flow");
    }
    else
    {
        printf(" enter a value to be
        pushed:");
        scanf("%d", &x);
        top++;
        stack[top] = x;
    }
}
```

WEEK - 2

28/12/23

Date
Page

Infix to Postfix Pgm

2a.

Write a pgm to convert an infix expression to postfix expression using stacks in C.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 20
char s[MAX];
int top = -1;
int prec(char c)
{
    if (c == '^')
        return 5;
    else if (c == '/')
        return 4;
    else if (c == '*')
        return 3;
    else if (c == '+')
        return 2;
    else if (c == '-')
        return 1;
    else
        return -1;
}
int isEmpty()
{
    return top == -1;
}
int isFull()
{
```

Date __/__/__
Page _____

```
    return top == MAX - 1;
}
char peek()
{
    return s[top];
}
char pop()
{
    if (isEmpty())
    {
        printf("Stack is empty\n");
        return -1;
    }
    else
    {
        char ch = s[top];
        top--;
        return ch;
    }
}
void push(char opr)
{
    if(isFull())
    {
        printf("Stack is full\n");
    }
    else
    {
        top = top+1;
        s[top] = opr;
    }
}
```

Date / /
Page

```
void infixToPostfix(char infix[], char postfix[])
{
    int i, j;
    char ch;
    for(i=0; j=0; i<strlen(infix); i++)
    {
        ch = infix[i];
        if((ch>='0' && ch<='9') || (ch>='A' && ch<='Z') || (ch>='a' && ch<='z'))
        {
            postfix[j++] = ch;
        }
        else if(ch==')')
        {
            while(!isEmpty() && peak()!='(')
            {
        else if(ch=='(')
        {
            push(ch);
        }
        else if(ch==')')
        {
            while(!isEmpty() && peak()!='(')
            {
                postfix[j++] = pop();
            }
            if(!isEmpty() && peak()!='(')
            {
                printf("Invalid expression\n");
                return;
            }
            else
```

Increments after assignment

```
        {
            pop();
        }
    }
    else
    {
        while (!isEmpty() && prec(ch)<=prec(peek()))
        {
            postfix[j++] = pop();
        }
        push(ch);
    } // end of else
} // end for loop
while (!isEmpty())
{
    postfix[j++] = pop();
}
postfix[j] = '\0';
} // end of infix to Postfix fn
int main()
{
    char infix[20], postfix[20];
    printf("Enter the infix expression
        string : \n");
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("Postfix expression : %s\n",
        postfix);
    return 0;
}
```

Output:

Enter the infix expression string:
a+(b/c)*d-(e^f)
Postfix expression : abc/d^+ef^-

26. Evaluation of postfix expression

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int n, top=-1, k;
int val, res, ans;
int stack[100];
char expn[100]
void push(int [], int);
int pop(int []);
int evaluate(char expn[]);

void main()
{
    printf("Enter a postfix expression:\n");
    scanf("%s", expn);
    ans = evaluate(expn);
    printf("Answer of above expression:\n");
    printf("%d", ans);
}
```

Date __/__/__
Page _____

```
int evaluate (char expn[])
{
    int op1, op2, i=0;
    n = strlen (expn);
    while(expn[i] != '\0')
    {
        if (isdigit (expn[i]))
            push (stack, (int) (expn[i] - 0));
        else
        {
            op1 = pop (stack);
            op2 = pop (stack);
            switch ( expn[i])
            {
                case '+' :
                    res = op1 + op2;
                    break;
                case '-' :
                    res = op1 - op2;
                    break;
                case '*' :
                    res = op1 * op2;
                    break;
                case '%' :
                    res = op2 % op1;
                    break;
                case '/' :
                    res = op2 / op1;
                    break;
                default :
                    continue;
}
```

```
            if (res < 0)
                res = res * (-1);
            push (stack, res);
        }
        i++;
    }
    return stack[top];
}

void push (int stack[], int val)
{
    if (top == n-1)
        printf ("Overflow");
    else
    {
        top += 1;
        stack [top] = val;
    }
}
int pop (int stack[])
{
    if (top == -1)
        printf ("Underflow");
    else
    {
        val = stack [top];
        top -= 1;
        return val;
    }
}
```

28/12/23

Date __/__/__
Page _____

OUTPUT:

Enter a postfix expression:
13*34*45-

Value: 9

WEEK-3 WEEK-3

3a. Queue Implementation using array

```
#include <stdio.h>
int q[50], rear = -1, front = -1, size;
void enqueue();
void dequeue();
void display();
void main()
{
    int ch;
    printf("Enter the size of queue: ");
    scanf("%d", &size);
    printf("Enter choice: ");
    printf("Press 1: Insert,
            2: delete  3: Display
            4: Exit \n");
    while(ch!=4)
    {
        printf("Enter choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue()
                break;
```

```
        case 3:
                display();
                break;
            }
        }
        printf("Exited");
}
void enqueue()
{
    int item;
    if (rear == size-1)
        printf("Queue is full\n");
    else
    {
        if (front == -1)
            front = 0;
        printf("Insert an element: ");
        scanf("%d", &item);
        rear += 1;
        q[rear] = item;
    }
}
void dequeue()
{
    if (front == -1 || front > rear)
        printf("Queue is empty\n");
    else
    {
        printf("Deleted element is: %d\n
            q[front]);
        front += 1;
    }
}
```

Date __/__/__
Page _____

```
void display()
{
    int i;
    if (front == -1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is \n");
        for(int i= front ; i <= rear; i++)
            printf("%d \t", q[i]);
        printf("\n");
    }
}
```

~~Circular~~ O/p:

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice
1
Enter the number to be inserted into the queue
2
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice
1
Enter the number to be inserted into the queue
3
1. Enqueue 2. Dequeue 3. Display 4. Exit.
2
2 was removed from the queue
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice : 3
Queue elements : 3

3b. Circular Queue Implementation

```
#include <stdio.h>
#include <stdlib.h>
#define size 50
int Q[size];
int rear = -1;
int front = -1;
int IsFull()
{
    if (front == (rear + 1) % size)
    {
        return 0;
    }
    else
    {
        return -1;
    }
}
int IsEmpty()
{
    if (front == -1 && rear == -1)
    {
        return 0;
    }
    else
    {
        return -1;
    }
}
```

Date __/__/__
Page _____

```
void Enqueue (int x)
{
    int item;
    if (IsFull () == 0)
    {
        printf (" Queue Overflow \n ");
        return ;
    }
    else
    {
        if (IsEmpty ()==0)
        {
            front = 0;
            rear = 0;
        }
        else
        {
            rear = (rear + 1) % size;
        }
        Q[rear] = x;
    }
}
int Dequeue ()
{
    int x;
    if (IsEmpty ()==0)
    {
        printf (" Queue underflow \n ");
    }
    else
    {
        if (front == rear)
        {
```

```
            x = q[front];
            front += -1;
            rear = -1;
        }
        else
        {
            x = q[front];
            front = (front + 1) % size;
        }
        return x;
    }
}
void Display()
{
    int i;
    if (IsEmpty() == 0)
    {
        printf("Queue is empty \n");
    }
    else
    {
        printf("Queue elements: \n");
        for (i = front; i != rear; i = (i+1) % size)
        {
            printf("%d \n", q[i]);
        }
        printf("%d \n", q[i]);
    }
}
```

```
void main()
{
    int choice, x, b;
    while(1)
    {
        printf("Enter the number to
        be inserted into the queue\n");
        printf("\t\t 1. Enqueue \t
                2. Dequeue \t
                3. Display \t
                4. Exit \n");
        printf("Enter your choice \n");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                printf("Enter the number
                to be inserted into the
                queue \n");
                scanf("%d", &x);
                Enqueue(x);
                break;
%d size)    case 2:
                b = Dequeue();
                printf("%d was removed
                from the queue \n", b);
                break;
            case 3:
                Display();
                break;
            case 4:
                exit(1);
            default:
```

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

O/P:

```
1. Enque 2. Dequeue 3. Display 4. Exit
Enter your choice
1
Enter the number to be inserted into the queue
10
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice
2
10 was removed from the queue
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice
3
Queue elements:
20
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice
4
```

Date __/__/__
Page _____

Linked List Implementation / Insertion

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
typedef struct Node
{
        int data;
        struct Node * next;
} Node;

Node *head = NULL;
void push();
void append();
void insert();
void display();
void main()
{
        int choice;
        while (1)
        {
            printf(" 1. Insert at beginning \t
                    2. Insert at end \t
                    3. Insert at position \t
                    4. Display \t
                    5. Exit \n ");
            printf(" Enter choice : \n ");
            scanf(" Enter choice : \n ");
            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");
        }
    }
}
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;
```

Date __/__/__
Page _____

```
        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", &new data);
        printf (" Enter position of new
                node : ");
        scanf ("%d", &pos);
        temp -> data = new data;
        temp -> next = NULL;
        if (pos == 0)
        {
            temp -> next = head;
            head = temp;
            return;
        }
        Node** *temp1 = head;
        while (pos--)
        {
```

```
            temp1 = temp1 -> next;
        }
        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\n");
}
```

O/P:

```
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: 10
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: 20
1. Insert at beginning 2. Insert at end
3. Insert at position 4. Display 5. Exit
Enter choice:
2
```

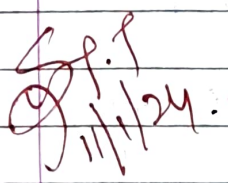
Date __/__/__
Page _____

Enter data in the new node : 32
Enter position of the new node : 2
1. Insert at the beginning 2. Insert at end
3. Insert at position 4. Display 5. Exit.
Enter choice
4

20 -> 10 -> 30 -> 32 -> NULL
1. Insert at beginning 2. Insert at end
3. Insert at position 4. Display 5. Exit.
Enter choice
5

Exiting the program.

);



18/1/24

Date / /
Page

WEEK-4

1. Linked list deletion & display fns

```
#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 pop (struct node **start);
void end_delete (struct node **start);
void delete_at_pos (struct node **start);
void free_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("\n 4. Delete a node from
        the end");
        printf("\n 5. Delete from a specific
        position");
        printf("\n 6: EXIT");
```

Date __/__/__
Page _____

```
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:
        pop(&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;
```

10 20 30 40 21

```
printf("Enter -1 to end\n");
printf("Enter the data:\n");
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 **start)
{
    if (*start == NULL)
    {
        printf("list is empty\n");
        return;
    }
    struct node *ptr = *start;
    struct node *ptr1 = NULL;
    while (ptr -> next != NULL)
    {
        ptr1 = ptr;
        ptr = ptr -> next;
    }
    if (ptr1 != NULL)
    {
```

```
            ptr1 -> next = NULL;
            free(ptr);
        }
        else
        {
            // Only one node in the list
            free(ptr);
            *start = NULL;
        }
    }
void delete_at_pos(struct node **start)
{
    if(*start == NULL)
    {
        printf("List is empty\n");
        return;
    }
    int loc;
    printf("\n Enter the location of the no
    which has to be deleted : ");
    scanf("%d", &loc);
    struct node *ptr = *start;
    struct node *ptr2 = NULL;
    for(int i=0; i<loc; i++)
    {
        ptr1 = ptr;
        ptr = ptr -> next;
        if(ptr==NULL)
        {
            printf("There are less than
            elements in the list\n", loc
            return;
        }
```

Date __/__/__
Page _____

```
if (ptr1 != NULL)
{
    ptr1 -> next = ptr -> next;
    free(ptr);
    printf(" Deleted node at
    %d position \n", loc);
}
else
{
    // Deleting the first node
    * start = ptr -> next;
    free(ptr);
    printf(" Deleted 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;
    }
}
```

%d
);

OUTPUT:

***** MAIN MENU *****

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

Enter your option: 1
Enter -1 to end
Enter the data:
10
Enter the data: 20
Enter the data: 30
Enter the data: 40
Enter the data: -1
LINKED LIST CREATED

***** MAIN MENU *****

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

Enter your option: 2

10 20 30 40

***** MAIN MENU *****

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

Date __/__/__
Page _____

Enter your option : 3
***** MAIN MENU *****
1 : Create a list
2 : Display the list
3 : Delete a node from the beginning
4 : Delete a node from the end
5 : Delete from a specific position
6 : EXIT
Enter your option : 2
20 30 40
Enter your option : 4 // deletes from end.
Enter your option : 2 // displays LL
20 30
Enter your option : 5 // delete from specific position
Enter the location of the node which has to be deleted : 1 // deletes element "30"
Deleted node at 1 position
~~Enter yo~~
Enter your option : 2 // ~~disp~~ displays
20
Enter your option : 6
Exiting.....

Sp.f
18/1/24

Week-5 Lab

2. MIN STACK Leet code Pgm

```
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(
    if (st -> s == NULL)
    {
        printf("memory allocation failed");
        free (st -> s);
        free (st -> minstack);
        exit(0);
    }
    return st;
}
```

Date __/__/__
Page _____

```
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];
        }
    }
}
```

```
);
* sizeof(int));
```

```
void minStackPop(MinStack * obj)
{
    int value;
    if (obj -> top == -1)
    {
        printf("underflow");
    }
    else
    {
        value = obj -> s[obj -> top];
```

```
        obj -> top--;
        printf("%d is popped\n", value);
    }
}
int minStackTop(MinStack* obj)
{
    int value = -1;
    if (obj -> top == -1)
    {
        printf("underflow\n");
        exit(0);
    }
    else
    {
        value = obj -> s[obj -> top];
        return value;
    }
}
int minStackGetMin(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);
}
```

Input

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

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

Stdout:

-3 is popped

Output:

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

Date
Page

Reversing a linked list

```
struct ListNode* reverseBetween(struct ListNode* head,
    int left, int right)
{
    if(head == NULL || left == right)
    {
        return head;
    }
    struct ListNode* dummy = (struct
    ListNode*)malloc(sizeof(struct ListNode));
    dummy -> next = head;
    struct ListNode* pre = dummy;
    // Move to the node just before the
    left position
    for(int i = 1; i < left; ++i)
    {
        pre = pre -> next;
    }
    // Reverse the nodes from left to right
    struct ListNode* current = pre -> next;
    struct ListNode* next = NULL;
    struct ListNode* prev = NULL;
    for(int i = 0; i <= right - left; ++i)
    {
        next = current -> next;
        current -> next = prev;
        prev = current;
        current = next;
    }
    // Connect the reversed position back
      to the original list
    pre -> next -> next = current;
```

1 2 3 4 5

Date __/__/__
Page _____

```
pre -> next = prev;
struct ListNode* result = dummy->next;
free(dummy);
return result;
}
```

* head,

Input:

head =

[1,2,3,4,5]

left =

2

right =

4

Output:

[1,4,3,2,5]

1 6 5 4 3 2 8

L R

1 2 3 4 5 6 7 8

Date
Page

WEEK - 6 DS

25/1/24

1. WAP to a) sort a singly LL
b) reverse a singly LL &
c) concatenate with another LL
in same pgm.

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node* next;
};
struct Node* head = NULL;
struct Node* head_ll2 = NULL;
// Fn to insert a node at the end
  of LL.

void insert_end (struct Node** h,
                 int data)
{
    struct Node* newNode = (struct Node*)
                malloc(sizeof(struct Node));
    newNode -> data = data;
    newNode -> next = NULL;
    if (*h == NULL)
    {
        *h = newNode;
    }
```

```
        else
        {
            struct Node* temp = *h;
            while(temp -> next != NULL)
            {
                temp = temp -> next;
            }
            temp -> next = newNode;
        }
}
void display(struct Node* h)
{
    struct Node* current = h;
    while(current != NULL)
    {
        printf("%d -> ", current -> data);
        current = current -> next;
    }
    printf("NULL\n");
}

void sort(struct Node *h)
{
    if(h == NULL || h -> next == NULL)
    {
        printf("List is already sorted\n");
        return;
    }
    int swapped;
    struct Node* ptr;
    do
    {
        swapped = 0;
```

```
        ptr = h;
        while (ptr -> next != NULL)
        {
            if (ptr -> data > ptr -> next -> data)
            {
                int temp = ptr -> data;
                ptr -> data = ptr -> next -> data;
                ptr -> next -> data = temp;
                swapped = 1;
            }
            ptr = ptr -> next;
        }
    } while (swapped);
}

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

void concatenate (struct Node ** list1,
                  struct Node * list2)
{
    if (* list1 == NULL)
```

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

int main()
{
    int choice;
    printf("\n MENU \n");
    printf("1. Sort the list \n");
    printf("2. Reverse the list \n");
    printf("3. Concatenate with another
      list \n");
    printf("4. Exit \n");
    printf("Enter your choice \n");
    scanf("%d", &choice);
    while(choice != 4)
    {
        switch(choice)
        {
            case 1:
                insert end(&head, 3);
                insert end(&head, 1);
                insert end(&head, 2);
```

Date __/__/__
Page

```
printf("Given
Linked List : \n");
display (head);
sort (head);
printf("\n Sorted List : \n");
display (head);
break;

case 2:
printf("Given Linked
list : \n");
display (head);
reverseList (& head);
printf("\n Reversed List: \n");
display (head);
break;

case 3:
insert_end (& head, 2, 5);
insert_end (& head, 2, 4);
printf("Given second
linked list : \n");
display (head, 2);
concatenate (& head, head, 2);
printf("\n Concatenated
List : \n");
display (head);
break;

case 4:
printf("Exiting ...");
break;

default;
printf("\n Invalid choice! \n");
}
printf("Enter your choice \n");
```

Date __/__/__
Page _____

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

OUTPUT:

```
MENU
1. sort the list
2. Reverse the list
3. Concatenate with another list
4. Exit
Enter your choice
1
Given linked list:
3 -> 1 -> 2 -> NULL
Sorted List:
1 -> 2 -> 3 -> NULL
Enter your choice:
2
Given linked list:
1 -> 2 -> 3 -> NULL
Reversed list:
3 -> 2 -> 1 -> NULL
Enter your choice:
3
Given second linked list:
5 -> 4 -> NULL
Concatenated list:
3 -> 2 -> 1 -> 5 -> 4 -> NULL
Enter your choice
4
Exiting....
```

Date
Page

2. WAP to implement Stacks using singly linked list

```
#include <stdio.h>
#include <stdlib.h>
// Define the Node structure
struct Node
{
    int data;
    struct Node* next;
};
// Declare the head of the LL
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\n");
        exit(1);
    }
    new_node -> data = new_data;
    new_node -> next = *head_ref;
    *head_ref = new_node;
    printf("%d pushed to the stack\n",
     new_data);
}
```

Date __/__/__
Page _____

```
void pop()
{
    struct Node * ptr;
    if (head == NULL)
    {
        printf("Stack is empty\n");
    }
    else
    {
        ptr = head;
        head = ptr -> next;
        printf("%d popped from the stack\n
        ", ptr -> data);
        free(ptr);
    }
}

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

P.T.O

```
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);
```

Date __/__/__
Page _____

```
    return 0;
}
```

O/P:

```
1. Push
2. Pop
3. Display
0. Exit
Enter your choice : 1
Enter data to push : 10
10 pushed to the stack

Enter your choice : 1
Enter data to push : 20
20 pushed to the stack
Enter your choice : 2
20 popped from the stack

Enter your choice : 3
 Stack : 10

Enter your choice : 0
Exiting program
```

delete at end

Date
Page

3. WAP to implement Queues using singly linked list

```
#include <stdlib.h>
#include <stdio.h>
struct Node
{
    int data;
    struct Node * next;
};
typedef struct Node Node;
Node* createNode(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;
```

Date __/__/__
Page _____

```
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 dequeuevalue = queue -> front -> data;
    Node *temp = queue -> front;
    queue -> front = queue -> front -> next;
    free (temp);
    return dequeuedvalue;
}
void main()
{
    LinkedList queue;
    queue.front = NULL;
    queue.rear = NULL;
    printf ("\n queue operations: \n");
```

```
enqueue (&queue, 40);
enqueue (&queue, 60);
enqueue (&queue, 80);
display (queue.front);
printf ("dequeued from queue:
%d \n", dequeue (&queue));
printf (" dequeued from queue:
%d \n", dequeue (&queue));
display (queue.front);
}
```

O/P:

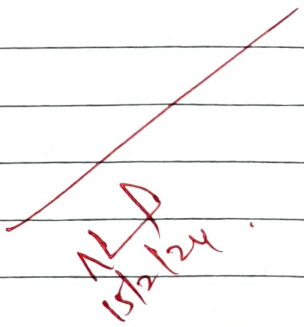
queue operations:

40 → 60 → 80 → NULL

dequeued from queue: 40

dequeued from queue: 60

80 → NULL



15/2/24

1/2/24

WEEK-7 DS Lab

Pgm 1: DOUBLY LINKED LIST IMPLEMENTATION

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

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

struct Node* createNode(int value)
{
    struct Node* newNode = (struct Node*) malloc
                           (sizeof(struct Node));
    if (newNode == NULL)
    {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode -> data = value;
    newNode -> prev = NULL;
    newNode -> next = NULL;
    return newNode;
}

void insertNodeToLeft(struct Node** head, struct
     Node* targetNode, int value)
{
    struct Node* newNode = createNode(value);
    if (targetNode -> prev != NULL)
    {
        targetNode -> prev -> next = newNode;
    }
    newNode -> prev = targetNode -> prev;
```

int *p = *q

```
        else
        {
            *head = newNode;
        }
        newNode -> next = targetNode;
        targetNode -> prev = newNode;
}

void deleteNodeByValue(struct Node** head, int value)
{
        struct Node* current = *head;
        while (current != NULL)
        {
            if (current -> data == value)
            {
                if (current -> prev != NULL)
                {
                    current -> prev -> next = current ->
                }
                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 in
            the list. \n", value);
}
```

o/p:

The initial linked list :

Doubly Linked List : 1 2 3

Inserting a new node to the left of 2nd node :

Doubly Linked List : 1 4 2 3

Enter the value whose corresponding node you want to delete :

2

Doubly Linked List : 1 4 3

Leetcode - ~~Doubly~~ Splitting a LL into k parts

```
struct ListNode** splitListToParts(struct ListNode
* head, int k, int* returnSize)
{
    int length = getLength(head);
    int partSize = length/k;
    int remainder = length%k;
    struct ListNode** result = (struct ListNode**)
                    malloc(k * sizeof(struct
    * returnSize = k;          ListNode*));
}
```

— X —

i=6 ?

Date __/__/__
Page: _____

WEEK - 8

Write a C program, given the head of a linked list, rotate the list to the right by k places

```
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;
    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;
}
```

Test Result :

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

k = 2

O/P:

[4, 5, 1, 2, 3]

Expected:

[4, 5, 1, 2, 3]

15/12/24

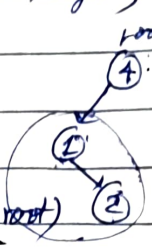
~~WEEK 7~~

1. Write a C pgm
   a. To construct a binary search tree
   b. To traverse the tree using all the methods i.e, in-order, preorder, postorder
   c. To display elements in the tree.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode
{
    int data;
    struct TreeNode * left;
    struct TreeNode * right;
};
struct TreeNode * createNode ( int data)
{
    struct TreeNode* newNode = (struct
        TreeNode*) malloc ( sizeof(struct TreeNode));
    newNode -> data = data;
    newNode -> left = NULL;
    newNode -> right = NULL;
    return newNode;
}
struct TreeNode * insert (struct TreeNode * root, int data)
{
    if(root == NULL)
        return createNode (data);
    if (data < root ->data)
        root -> left = insert (root -> left, data);
    else if ( data > root ->data)
```

Date __/__/__
Page _____

```
        root->right = insert(root->right, data);
        return root;
    }
void inorderTraversal(struct TreeNode* root)
{
    if(root!=NULL)
    {
        inorderTraversal(root->left);
        printf("%d", root->data);
        inorderTraversal(root->right);
    }
}
void preorderTraversal(struct TreeNode* root)
{
    if(root!=NULL)
    {
        printf("%d", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}
void postorderTraversal(struct TreeNode* root)
{
    if(root!=NULL)
    {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d", root->data);
    }
}
```



4 1 2

1 2 4

2 1 4

P.T.O

```
void display(struct TreeNode * root)
{
    printf("In-order Traversal: ");
    inorderTraversal(root);
    printf("\n Pre-order Traversal: ");
    preorderTraversed(root);
    printf("\n Post-order Traversal: ");
    postorderTraversal(root);
    printf("\n");
}

int main()
{
    struct TreeNode * root = NULL;
    int num_root;
    int num;
    printf("Enter the root node data \n");
    scanf("%d", &num_root);
    root = insert(root, num_root);
    printf("Enter -1 to end \n");
    printf("Enter data for each node \n");
    scanf("%d", &num);
    while(num != -1)
    {
        insert(root, num);
        printf("Enter the data \n");
        scanf("%d", &num);
    }
    display(root);
    return 0;
}
```

DS Lab internals

Date / / 7th March

Page

O/P:

Enter the root node data

10

Enter -1 to end

Enter data for each node

8

Enter the data

20

Enter the data

15

Enter the data

30

Enter the data

2

Enter the data

9

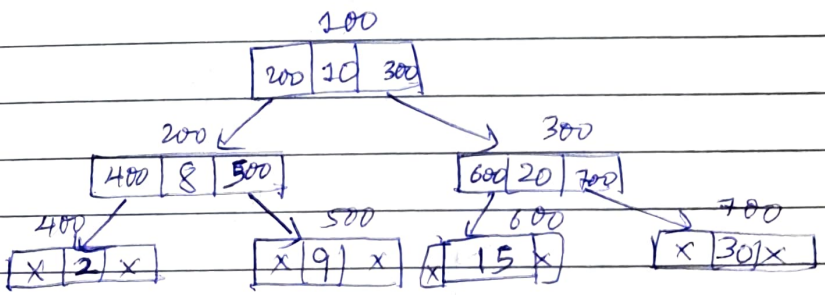
Enter the data

-1

In-order traversal: 2 8 9 10 15 20 30

Pre-order traversal: 10 8 2 9 20 15 30

Post-order traversal: 2 9 8 15 30 20 10



OOT Lab:

Wed - 9 &

10

NOT THERE

DS Observation - Book - 2 USN : 1BM22CS207

INDEX

(WEEK-8)

DS ~~Week~~

NAME: Preethi N STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: _____

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	22/2/24	Week-9 a- Traverse a tree with BFS b. with DFS		Sign.
2.	29/2/24	Week-10 a- Perform Hashing with linear Probe b. Hackerrank: Swapping nodes in a tree		

WEEK-8 (22/2/24)

1. Write a C program to traverse a graph using BFS method.

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

struct Queue{
    int items[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][dest] = 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++)
        {
            if(graph->adjMatrix[currentVertex][i]
                && !visited[i])
            {
                visited[i] = true;
            }
        }
    }
}
```

```
int main()
{
    struct Graph graph;
    int vertices, edges, startVertex;
    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);
    createGraph(&graph, 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: ", i++);
        scanf("%d %d", &src, &dest);
        addEdge(&graph, src, dest);
    }
    printf("Enter the starting vertex: ");
    scanf("%d", &startVertex);
    printf("BFS Traversal:");
    BFS(&graph, startVertex);
    return 0;
}
```

OUTPUT:

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

Enter the starting vertex : 0

BFS Traversal : 0 1 2 3 4 5

DFS Traversal

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_SIZE 100
struct Graph {
        int vertices;
        bool adjMatrix[MAX_SIZE][MAX_SIZE];
        bool visited[MAX_SIZE];
};
void createGraph(struct Graph* graph, int vertices)
{
        graph->vertices = vertices;
        for(int i=0; i< vertices ; i++)
        {
                graph->visited[i] = false;
                for(int j=0; j< vertices; j++)
                {
                        graph->adjMatrix[i][j] = false;
                }
        }
}
void addEdge(struct Graph* graph, int src, int dest)
{
        graph->adjMatrix[src][dest] = true;
        graph->adjMatrix[dest][src] = true;
}
void DFS(struct Graph* graph, int vertex, int* count)
{
        (*count)++;
        graph->visited[vertex] = true;
```

```
            graph -> adjMatrix[vertex][i] ...
          {         visited[i])
                      DFS(graph, i, count);
}
          }       }
int
{   main()

    struct graph graph;
    int vertices, edges;
    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    createGraph(&graph, vertices);
    printf("Enter number of edges: ");
    scanf("%d", &edges);
    for (int i=0; i<edges; i++)
    {
        int src, dest;
        printf("Enter edge %d source and destination : ", i+1);
        scanf("%d %d", &src, &dest);
        addEdge(&graph, src, dest);
    }
    int count = 0;
    printf("DFS Traversal");
    DFS(&graph, 0, &count);
    if (count == vertices)
        printf("\n graph is connected.\n");
    else
        printf("\n graph is not connected.\n"
```

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

int main()
{
    struct Graph graph;
    int vertices, edges;
    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    createGraph(&graph, vertices);
    printf("Enter number of edges: ");
    scanf("%d", &edges);
    for (int i = 0; i < edges; i++)
    {
        int src, dest;
        printf("Enter edge %d source and
        destination: ", i+1);
        scanf("%d %d", &src, &dest);
        addEdge(&graph, src, dest);
    }
    int count = 0;
    printf("DFS Traversal");
    DFS(&graph, 0, &count);
    if (count == vertices)
        printf("\n Graph is connected.\n");
    else
        printf("\n Graph is not connected.\n");
    return 0;
}
```

Q1:

Enter the number of vertices and edges: 5 4 3

Enter the edges (in the format 'source destination'):

0 1

2 3

1 3

1 4

3 4

0 4

0 2

~~Enter the starting vertex:~~

Enter edge 1 source and destination: 0 1

Enter edge 2 source and destination: 1 2

Enter edge 3 source and destination: 0 2

DFS Traversal:

Graph is not connected.

29/2/24

Week - 10 Lab

~~Week-9 Lab~~

write a C program to perform hashing with linear probe

```
#include <stdio.h>
#define TABLE_SIZE 10
int hashTable[TABLE_SIZE];
void initializeHashTable()
{
    for(int i = 0; i < TABLE_SIZE; i++)
    {
        hashTable[i] = -1;
    }
}
void insert(int index, int key)
{
    if(index < 0 || index >= TABLE_SIZE)
    {
        printf("Invalid index.\n");
        return;
    }
    if(hashTable[index] == -1)
    {
        // if slot is empty insert the key
        hashTable[index] = key;
        printf("Inserted %d at index %d\n",
        key, index);
    }
    else
    {
        printf("Unable to insert %d at index
        %d. Slot is occupied.\n", key, index);
    }
}
```

```
int search (int key)
{
    int i = 0;
    int hkey = key % TABLE_SIZE;
    int index;
    do
    {
        index = (hkey + i) % TABLE_SIZE;
        if (hashTable[index] == key)
        {
            // If element at that index is the
               search value, print element found
               and stop
            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()
{
    initializeHashTable();
    int index, value;
    printf ("Enter index and value to insert
      (-1 to stop): \n");
    do
    {
        scanf ("%d %d", &index, &value);
        if (index != -1 && value != -1)
            insert (index, value);
    } while (index != -1 && value != -1);
    printf ("Enter values to search (-1 to stop): \n");
```

```
    do
    {
        scanf("%d", &value);
        if (value != -1)
            search(value);
    } while (value != -1);
    return 0;
}
```

OUTPUT:

```
Enter index and value to insert (-1 to stop):
1 10
Inserted 10 at index 1
2 20
Inserted 20 at index 2
3 30
Inserted 30 at index 3
-1 -1
Enter values to search (-1 to stop):
10
Element 10 found at index 1
-1
Process returned (Program completed)
```

Hackerrank

Swapping the nodes in the tree

```
#include <assert.h>
#include <stdbool.h>
#include <stdio.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;
    ~~return;~~
    inOrderTraversal (root -> left, result, index);
    result [(*index)++] = root -> data;
    inOrderTraversal (root -> right, result, index);
}

void swapAtLevel (Node* root, int k, int level)
{
    if (root == NULL) return;
    if (level % k == 0)
    {
        Node* temp = root -> left;
        root -> left = root -> right;
```

```
        root -> right = temp;
    }
    swapAtLevel(root -> left, k, level + 1);
    swapAtLevel(root -> right, k, level + 1);
}
```

Sol: 04/03/24