

I N D E X

Name P. Manya Std 3 Sec 3-D  
Roll No. \_\_\_\_\_ Subject DS Lab School/College \_\_\_\_\_

Sl. No.	Date	Title	Page No.	Teacher Sign/ Remarks
1.	7/12/23	Lab - 1	10	Ys
2.	21/12/23	Lab - 2	10	Ys
3.	28/12/23	LAB - 3	10	Ys
4.	20/1/24	LAB - 4	10	Ys
5.	11/1/24	LAB - 5	10	S
6.	18/1/24	LAB - 6.	10	S
7.	18/1/24	Leet code : 2	10	S
8.	11/1/24	Leet code : 1	5	S
9.	25/1/24	Lab - 7	10	S
10.	1/2/24	LAB - 8	10	Ys
11.	1/2/24	leet code : Split linked list	10	Ys
12.	15/2/24	Lab - 9	10	Ys
13.	18/2/24	leet code : linked list	10	Ys
14.	22/2/24	Mark as mark	10	- W
15.	29/2/24	Lab 10 :	10	- Ok

8 practice programs outputs:

1. Enter your name

MARYA P

Enter account number

98765

Enter amount

100000000

Enter amount to be withdrawn

3456780

Remaining balance = 96543220

Enter amount to be deposited

10000

Balance = 96553220

2. Enter 5 words: END

Boring

MARYA

NECKlace

hungry

In lexicographical order

Boring

MARYA

NECKlace

END

hungry

3. Enter the no of elements in the array: 3

Enter the elements: 111

222

333

Enter no to be searched = 222

222 is found at position = 2

4. Happy go lucky personality this str  
contains 'luck'.

5. Enter size of array : 5

Enter the elements of the array :

Element 1 : 4

Element 2 : 7

Element 3 : 2

Element 4 : 7 is at position 3

Element 5 : 9

Enter 8 means to find the last occurrence

Last occurrence of 7 is at position 4

The last occurrence of 7 is at index

6. Enter no. of elements in array : 5

Enter elements : 890

876

567

867

874

Enter no. to be searched : 867

867 is found at position = 4

7. Enter the no. of elements in array : 9

Enter the elements : 12

13

23

24

35

36

47

48

59

Enter the no that has to be searched : 60  
60 DOES NOT EXIST IN THE ARRAY

3. Enter size of the array : 3

Enter elements in the array : 12 34 56 78  
67 34 45 48 50

9 8 5 5 3 7 7 3 8

maximum element = 9 8 5 5 3 7 7 3 8

minimum element = 12 34 56 78

(not in the array)

not in the array

not in the array

minimum

maximum

minimum, maximum

("maximum first") finding

(maximum, "b", "I found")

("an element not") finding

b = minimum: minimum, "a", "I found")

(minimum, "a", "b", "I found")

(minimum, "a", "b", "I found")

b = maximum: maximum, "b", "I found")

(maximum, "b", "a", "I found")

maximum

minimum

maximum

maximum, "b", "I found")

minimum, "a", "I found")

minimum, "a", "b", "I found")

maximum, "b", "a", "I found")

LAB 2nd id of var don't change after swap  
VARAIS SITE IN PLEX TALK SOON

Pgm - 1

Swapping of 2 nos 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 number : ");
```

```
scanf ("%d", &num1);
```

```
printf ("Enter second no ");
```

```
scanf ("Before swapping : num1 = %d, num2 = %d",
```

```
&num1, &num2);
```

```
swap (&num1, &num2);
```

```
printf ("After swapping : num1 = %d,
```

```
num2 = %d", num1, num2)
```

```
return 0;
```

```
}
```

Output:

Enter 1<sup>st</sup> number : 12

Enter second number : 24

Before swapping : num1 = 12, num2 = 24

After swapping : num1 = 24, num2 = 12

2. Pgmr - 2

# include &lt;stdio.h&gt;

# include &lt;stdlib.h&gt;

void \* mymalloc (size\_t size);

{ return malloc (size); }

void realloc (void \* pto, size\_t size);

{ return malloc (size); }

{ void \* realloc (size\_t num; size\_t size) { }

{ + return realloc (num, size); }

void free (void \* pto);

int main () {

int \* arr, \* arr2;

size\_t size;

printf (" Enter size ");

scanf ("%d", &amp; size);

arr = (int \*) malloc (size \* sizeof (int));

if (arr == NULL)

printf (" Memory allocation failed ");

return 1;

}

printf (" Enter elements ");

for (size\_t i = 0; i &lt; size; i++)

printf (" Elements %2n " "%d ");

scanf ("%d", &amp; arr[i]);

printf (" Elements %d ", arr[0]);

for (size\_t i = 0; i &lt; size; i++)

```
print("%.d", arr[i]);
```

```
printf("ln");
```

$$\sin \theta = 2;$$

an<sup>o</sup> 2 := (int) realse  
if Carr 2 =: NULL)

printf("Memory allocation failed.\n")

Fri. Corr. D)

3

printf("Enter additional elements (%d): ");

for ( i < j ; i = single[2] , j = single[1] )  
print( " Element " , i+1 );

2

"printf" Elements of array (recall: !\n for line t. i=0; )

```
printf("%d", arr2[i]);
```

```
printf("in");
```

for "car 2"); 12, "v" for  
"action 2"; 13 ("fin") - 14

Chennai  
Antennas

Enter size of array : 2  
Enter elements :

## Element 2: 24

Elements of fairplay (maths):

Enter additional elements:  Element 3: 3b

Element 3: 3k

Element 4: 90  
 Elements: 12 24 36 90

3. Stack Implementation

```

#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 10
stack stack {
    int items[MAXSIZE];
    int top;
}
void initialise(stack *stack) {
    stack->top = -1;
}
int isEmpty(stack *stack) {
    return stack->top == -1;
}
int isFull(stack *stack) {
    return stack->top == MAXSIZE - 1;
}
void push(stack *stack, int value) {
    if (isFull(stack)) {
        printf("Stack overflow. Can't push %d.\n", value);
    } else {
        stack->top++;
        stack->items[stack->top] = value;
        printf("Pushed %d.\n", value);
    }
}
int pop (stack *stack) {
}
  
```

```
if (!isEmpty(stack)) { OP: N times
    printf("Stack is empty.");
}
else {
    printf("Elements in stack:"); d
    for (int i=0; i<=stack->top; i++) OP: O(N)
        printf("%d", stack->items[i]); d
    printf("\n"); d
}
int main() XAM: number of
{
    struct stack *stack;
    initialize(&stack); d
    push(&stack, 10); d
    push(&stack, 20); d
    push(&stack, 30); d
    display(&stack); d
    pop(&stack); d
    display(&stack); d
    push(&stack, 40); d
    display(&stack); d
    return 0; d
}
Output: Pushed 10 into stack
Pushed 20 onto stack: 10 20 < stack
Pushed 30 onto stack: 10 20 30 < stack
Elements in stack: 10 20 30 < stack
Popped 20 from stack: 10 30 < stack
Elements in stack: 10 30 < stack
```

Pushed 10 into stack  
Pushed 20 onto stack: 10 20 < stack  
Pushed 30 onto stack: 10 20 30 < stack  
Elements in stack: 10 20 30 < stack  
Popped 20 from stack: 10 30 < stack  
Elements in stack: 10 30 < stack

Pushed 40 onto stack

Elements in stack: 10 20 40

~~Reha~~

21) 12/23

12/23 11/16

12/23 11/16

12/23 11/16

12/23 11/16

(12/23 11/16)

(12/23 11/16)

(12/23 11/16)

(12/23 11/16)

(12/23 11/16)

(12/23 11/16)

(12/23 11/16)

(12/23 11/16)

(12/23 11/16)

(12/23 11/16)

28/12/23

## WEEK - 2

1.

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50
char stack[SIZE];
int top = -1;
push(char ch)
```

```
{ stack[++top] = ch;
```

3

```
char pop()
```

{

```
return stack[top--];
```

}

```
int po(char symbol)
```

{

```
if (symbol == '+' ||
```

}

```
action(3);
```

}

```
else if (symbol == '*' ||
```

}

```
action(2);
```

}

```
else if (symbol == '+' ||
```

}

```
action(1);
```

}

```
else
```

```
action(0); } }
```

void main ()

```

    {
        char infix[50], postfix[50], ch;
        int i=0, k=0;
        printf("Enter infix expression");
        scanf("%s", infix);
        push('#');
        while ((ch = infix[i++]) != '#') {
            if (ch == '(') {
                push(ch);
            } else if (isalnum(ch)) {
                postfix[k++] = ch;
            } else if (ch == ')') {
                ch = pop();
                while (stack[top] != '(') {
                    postfix[k++] = pop();
                }
            }
        }
        while (pop(stack[top])) {
            postfix[k++] = pop();
        }
        printf("\nPostfix expression is %s\n", postfix);
    }
}

```

Output:

Entire infix expression =  $A * B + C * D - E$

Postfix expression =  $AB * CD * + E -$

## 2. Evaluation of Postfix

#include <stdio.h>

#include <ctype.h>

int stack[20];

void push(int x)

stack[++top] = x;

int pop() { ++top; }

return stack[top--];

int main()

char exp[20];

char \*c;

int n1, n2, n3, num;

printf("Enter the expression: ");

scanf("%s", exp);

c = exp; /\* pointer to first character \*/

while (\*c != '\0')

{ if (is digit (\*c))

num = \*c - '0';

```
push (num);  
}
```

18

*(L.)* *schulzii* # 108.

02 XAM 11/10/18

$w_1 = \text{pop}(2, i)$  *new word line*  
 $w_2 = \text{pop}(1, i)$  *it = new line*  
switch (\*i) *i++ = new line*

```
    n3 = n1+n2; // sum of first two numbers  
    break; // exit from loop  
} // end of loop
```

carl ' - ' ;  
i (will in many "Mistress  
of the house";

(5.i) break; ~~break~~; ("w/") it will

case "1";  
} C:\inetpub\wwwroot

$n_3 = n_1/n_2$ ; break; sinks stir

sunsh (n<sup>3</sup>); 3 (1) W. J. S.

~~printf(" %c\n", i);~~ // invalid expression

✓  $\text{exp}(\text{pop})$ ;  
:  $\text{pop} \cdot \text{exp}$ .

Antwerp ("Anvers" in French) is a port city in Belgium.

Enter the expression: 4.5 \* 2.1

Result of expression is.  $1.5 * 9^4 = 29$

## LAB - 3

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

```
#define MAX 50
```

```
int queue[0:MAX];
```

```
int rear = -1; i (Index = -1)
```

```
int front = -1; (i * ) i (Index = -1)
```

```
display()
```

```
int i;
```

```
if (front == -1) S.N + 1 = S.N
```

```
printf(" Queue is full ");
```

```
else
```

```
printf(" Queue is ");
```

```
for (i = front; i <= rear; i++)
```

```
printf("%d", queue[i]);
```

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

```
main()
```

```
int choice;
```

```
while (1)
```

```
printf(" 1. Insert ");
```

```
printf(" 2. Delete ");
```

```
printf(" 3. Display ");
```

```
printf(" 4. Exit ");
```

```
printf(" Enter your choice ");
```

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

```
switch (choice)
```

```
case 1:
```

insert() // inserting element at index 7 front

break;

case 2:

delete();

break;

case 3:

display();

break;

printf("invalid character");

33 y

insert()

{ // inserting element at front

int add\_item();

if (rear == MAX - 1)

printf("queue overflow");

else

{ // inserting element at front

if (front == -1)

front = 0;

printf("insert the element");

scanf("%d", &add\_item);

rear += 1;

queue array[rear] = add\_item;

33 { // inserting element at front

delete();

{

if (front == -1 || front > rear)

printf("queue underflow");

return;

}

else

printf ("deleted element in ' %d ' given  
[front] ) ;

front += 1 ;

Output

1. insert
2. delete
3. display
4. exit

enter your choice : 1

Insert the element in queue = 3

1. insert
2. delete
3. display
4. exit

enter your choice : 1

insert element in queue : 3

1. insert
2. delete
3. display
4. exit

enter your choice : 2

deleted element is : 3

1. insert
2. delete
3. display
4. exit

enter your choice : 3

queue is :

## 3.b. Circular Queue

# include &lt; stdio.h &gt;

# define SIZE 5

int items [SIZE];

int front = -1, rear = -1;

int isFull () {

if ((front == rear + 1) || (front == 0 && rear  
== size - 1))

return 1; /\* (It true) = true \*/

return 0;

} /\* big found. with "1" finding

int isEmpty () {

if (front == -1)

return 1;

return 0;

} /\* small box

void enqueue (int element) {

if (isFull)

printf ("In Queue is full");

else if (rear &lt; front) /\*

if (front == -1) \*/

front = 0, rear = 0, item = 0;

rear = (front + 1) % SIZE;

items [rear] = element;

printf ("In Inserted = '%d', element);

}

int deQueue () {

int element;

if (isEmpty ())

printf ("In Queue is empty");

return -1;

}

else {

element = items[front];

if (front == rear)

front = -1;

rear = -1;

}

} else if (front &lt; rear) {

front = (front + 1) % SIZE;

rear = (rear + 1) % SIZE;

}

printf(" Delete element -&gt; %d \n")

action element; front in line;

}

void display() {

int i;

if (!is\_empty())

printf(" Empty Queue () \n");

printf(" Front -&gt; %d \n", front);

printf(" items -&gt; ");

for (i = front; i = rear; i = (i + 1) % SIZE) {

printf("%d ", items[i]);

printf("\n");

printf(" Rear -&gt; %d \n", rear);

}

int main() {

enqueue(1);

enqueue(2);

enqueue(3);

enQuene (4);  
 enQuene (5);  
 display ();  
 deQuene ();  
 deQuene (Y);  
 display ();  
 enQuene (6);  
 enQuene (7);  
 display ();  
 remove();  
 }  
 Insert = off \* short elements  
 } (1) push first line

Output: 3 (11111 = off ) fine  
 (enters in tail ) third

inserted 1  
 inserted 2  
 inserted 3 (" " no elements ) " ) third  
 inserted 4 (11111 = off ) fine,  
 inserted 5 (enters & off , " b " ) third  
 front - 0  
 items - 12345  
 rear 4

Deleted element off \* short elements

Deleted element off \* 2 are deleted off of next  
 item

front 2 ~~one id of index what ) third~~  
 items 345 ~~(enters & off , " b " ) front~~

rear 4 ~~insert = from & off~~

inserted 6 ~~from = head~~

inserted 7 ~~from = head~~

front 2 ~~one id of index what ) third~~

items 34567 ~~with \* short elements~~

rear 1

11/11/24

LAB - 4

## SINGLY LINKED LIST

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

struct node *head = NULL;

void display() {
    struct node *ptr = head;
    if (ptr == NULL) {
        printf("list is empty");
        return;
    }
    printf("elements are ");
    while (ptr != NULL) {
        printf("%d", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}

void insert_begin() {
    struct node *temp;
    temp = (struct node *)malloc(sizeof(struct node));
    printf("Enter value to be inserted");
    scanf("%d", &temp->data);
    temp->next = head;
    head = temp;
}

void insert_end() {
    struct node *temp1, *ptr;
    temp1 = (struct node *)malloc(sizeof(struct node));
    printf("Enter value to be inserted");
    scanf("%d", &temp1->data);
    ptr = head;
    while (ptr->next != NULL)
        ptr = ptr->next;
    ptr->next = temp1;
}
```

```
temp = (Struct node *) malloc (size of Struct  
node);  
printf("Enter value");  
scanf("%d", &temp->data);  
temp->next = NULL;  
if (head == NULL)  
    head = temp;  
else {  
    ptr = head;  
    while (ptr->next != NULL) {  
        ptr = ptr->next;  
    }  
    ptr->next = temp;  
}  
void insert_pos()  
{  
    int pos, i;  
    Struct node * temp, * ptr;  
    temp = (Struct node *) malloc (size of Struct  
node);  
    printf("Enter position");  
    scanf("%d", &pos);  
    printf("Enter value");  
    scanf("%d", &temp->data);  
    temp->next = NULL;  
    if (pos == 0)  
        temp->next = head;  
    else {  
        head = temp;  
        for (i=0; i<pos-1; i++) {  
            ptr = ptr->next;  
        }  
    }  
}
```

```

if (ptr == NULL) {
    printf("Position not found");
}
ptr = temp;
ptr->next = ptr->next;
ptr->next = temp;
}

```

```
int main() {  
    int choice; // variable declaration  
    while(1) {  
        printf("1. Insert at beginning, 2. Insert  
        at end 3. Insert at any position,  
        4. Display, 5. Exit");  
        scanf("%d", &choice);  
        if (choice == 1) {  
            insertAtBeginning();  
        } else if (choice == 2) {  
            insertAtEnd();  
        } else if (choice == 3) {  
            insertAtAnyPosition();  
        } else if (choice == 4) {  
            display();  
        } else if (choice == 5) {  
            exit(0);  
        } else {  
            printf("Invalid choice");  
        }  
    }  
}
```

Case 2:  $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$

*missed mid-11*

~~break is~~

~~case 3:~~

~~case 3 : Staff - der Edemat~~

visit - post);  
break;

break /breɪk/ T 384 S 4

Case 4: *labeled*

double 12' i want : blue

display();  
break;

*break;*

Page 5:

$\lim_{x \rightarrow 0} f(x)$  does not exist.

$\text{Ex. } t(10)$ ; break;

default:

```
printf("Enter correct choice");
```

}

```
return 0;
```

}

Output:

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

Enter your choice : 1

Enter the value to be inserted : 9

8: enter return

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

Enter your choice : 1

Enter value : 5

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

Enter your choice : 2

Enter value : 6

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

Enter choice : 4

Elements are 5 9 6

1. Insert at beginning to front
2. Insert at end from back to front
3. Insert at position move to front
4. Display
5. Exit

Enter your choice : 3

Enter position : 2

Enter value : 8

1. Insert at beginning to front
2. Insert at end from back to front
3. Insert at any position
4. Display
5. Exit

Enter choice : 4

Elements are 5 9 8 6 : value added

1. Insert at beginning to front
2. Insert at end from back to front
3. Insert at any position to front
4. Display
5. Exit

Enter your choice : 5

Final list

11/1/24 LEET CODE: 1: MIN STACK

#include <stdlib.h>

typedef struct {

int \* stack;

int \* minStack;

int top;

} Minstack;

Minstack\* minStackCreate() {

Minstack\* stack = (Minstack\*) malloc (sizeof(Minstack));

stack->stack = (int\*) malloc (sizeof(int)\*10000);

stack->minStack = (int\*) malloc (sizeof(int)\*10000);

stack->top = 1;

obj->top += 1;

obj->stack [obj->top] = val;

if (obj->top == 0) minStack->minStack [obj->top] = val;

else minStack->minStack [obj->top] = obj->minStack

[obj->top - 1];

mid minStackPop(Minstack\* obj) {

obj->top --;

int minstackTop(minstack\* obj)

{  
return obj->stack [obj->top];}

int minstackGetMin(minstack\* obj)

{  
return obj->minStack [obj->top];}

void minstackFree(minstack\* obj)

{  
free (obj->stack);

free (obj->minStack);

free (obj);

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

[[], [-2], [0], [-3], [1], [2] if [1, 1]]

~~Output:~~ [null, null, null, null, -3, null, 0, -2]

~~Expected:~~ [null, null, null, null, -3, null, 0, -2].

18/11/24

## LAB PROGRAM: 5 SINGLY LINKED LIST

```

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

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

struct node * head = NULL; // Line 1

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

void insert_begin() {
    struct node temp = (struct node) malloc(sizeof(struct node));
    printf("Enter value to insert: ");
    scanf("%d", &temp->data);
    temp->next = head;
    head = temp;
}

void delete_begin() {
    if(head == NULL) {
        printf("List is empty");
    }
}

```

```

3 printf("list empty"); 3 MAI 2029 DAJ
    return;
}
struct node *temp = head;
head = head->next;
printf("Element deleted");
free(temp);
}

void deleteNode()
{
    if (head == NULL)
        printf("list empty");
    else
    {
        struct node *temp, *prev;
        temp = head;
        while (temp->next != NULL)
        {
            prev = temp;
            temp = temp->next;
        }
        prev->next = NULL;
        if (temp == head)
            head = temp->next;
        else
            head = head->next;
        free(temp);
        printf("Element deleted, temp->data");
    }
}

```

```
3 void delete_at_position()
{
    int position;
    printf("Enter the position to delete: ");
    scanf("%d", &position);

    if (head == NULL)
    {
        printf("List is empty");
        return;
    }

    struct node *temp, *prev;
    temp = head;

    if (position == 0)
    {
        head = head->next;
        printf("Element at position %d deleted", position);
        free(temp);
        return;
    }

    for (int i=0; temp != NULL && i < position; i++)
    {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL)
    {
        printf("Position is out of bounds", position);
        return;
    }

    prev->next = temp->next;
    printf("Element deleted", position);
}
```

```
tree(temp);
int main()
{
```

    while(1)

        1. Enter choice of insertion or deletion  
        2. Insert at beginning, 3. delete at beginning, 4. delete at end, 5. delete at any position, 6. exit (n);

    printf("Enter choice:");  
    scanf("%d", &choice);  
    switch (choice)

    {  
        case 1:  
        {  
            if (root == NULL)  
                insert\_begin();  
            break;

    Case 2:  
    {  
        if (root == NULL)  
            delete\_begin();  
        break;

    Case 3:

    {  
        delete\_end();  
        break;

    Case 4:  
    {  
        delete\_at\_pos();  
        break;

    Case 5:  
    {  
        display();  
        break;

    Case 6:  
    {  
        exit(0);  
        break;

    default:  
    {  
        printf("Enter correct choice\n");  
        break;

    };  
    y

antpt:

1. insert at beginning
2. to delete at beg
3. delete at end
4. delete at position
5. display
6. exit

1 : insert at beg  
Enter your choice : 11 enter at ...  
Enter value to insert : 11

1. insert at beginning
2. to delete at beg
3. delete at end
4. delete at position
5. display
6. exit

1 : insert at beg  
Enter your choice : 1  
Enter value to insert : 12

- ~~1. insert at beg
  2. delete at beg
  3. delete at end
  4. delete at position
  5. display
  6. exit~~

Enter choice : 1

Enter value to insert : 13

- ~~1. insert at beg
  2. insert at beg
  3. delete at end
  4. delete at position
  5. display
  6. exit~~

Enter value: 14

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

11.

12.

13.

14.

15.

16.

17.

18.

19.

20.

21.

22.

23.

24.

25.

26.

27.

28.

29.

30.

31.

32.

33.

34.

35.

36.

37.

38.

39.

40.

41.

42.

43.

44.

45.

46.

47.

48.

49.

50.

51.

52.

53.

54.

55.

56.

57.

58.

59.

60.

61.

62.

63.

64.

65.

66.

67.

68.

69.

70.

71.

72.

73.

74.

75.

76.

77.

78.

79.

80.

81.

82.

83.

84.

85.

86.

87.

88.

89.

90.

91.

92.

93.

94.

95.

96.

97.

98.

99.

100.

101.

102.

103.

104.

105.

106.

107.

108.

109.

110.

111.

112.

113.

114.

115.

116.

117.

118.

119.

120.

121.

122.

123.

124.

125.

126.

127.

128.

129.

130.

131.

132.

133.

134.

135.

136.

137.

138.

139.

140.

141.

142.

143.

144.

145.

146.

147.

148.

149.

150.

151.

152.

153.

154.

155.

156.

157.

158.

159.

160.

161.

162.

163.

164.

165.

166.

167.

168.

169.

170.

171.

172.

173.

174.

175.

176.

177.

178.

179.

180.

181.

182.

183.

184.

185.

186.

187.

188.

189.

190.

191.

192.

193.

194.

195.

196.

197.

198.

199.

200.

201.

202.

203.

204.

205.

206.

207.

208.

209.

210.

211.

212.

213.

214.

215.

216.

217.

218.

219.

220.

221.

222.

223.

224.

225.

226.

227.

228.

229.

230.

231.

232.

233.

234.

235.

236.

237.

238.

239.

240.

241.

242.

243.

244.

245.

246.

247.

248.

249.

250.

251.

252.

253.

254.

255.

256.

257.

258.

259.

260.

261.

262.

263.

264.

265.

266.

267.

268.

269.

270.

271.

272.

273.

274.

275.

276.

277.

278.

279.

280.

281.

282.

283.

284.

285.

286.

287.

288.

289.

290.

291.

292.

293.

294.

295.

296.

297.

298.

299.

300.

301.

302.

303.

304.

305.

306.

307.

308.

309.

310.

311.

312.

313.

314.

315.

316.

317.

318.

319.

320.

321.

322.

323.

324.

325.

326.

327.

328.

329.</p

Element at position 2 deleted

1.

Elements moved upward & shifted down  
( $i = i + 1$ ,  $j = j - 1$ )

2.

3.

4.

5.

6.

Enter choice: 5 now & shift in down

Elements all move  $\rightarrow$  1  $\rightarrow$  NULL up & down

1.

2.

3.

4.

5.

6.

~~Enter choice: 6~~

$i = 0$  for

(JUNK : 1st) down

$j = 0$  for

(1 - 0 = 1) for

$i = 0$  for

(0 = 0) for

$i = 0$  for

(0 = 0) for

$i = 0$  for

(0 = 0) for

$i = 0$  for

input & output = 0 & shift in down

$i = 0$  for

(JUNK : 1st) down

(1 - 0 = 1) for

18/1/24 LEET CODE: singly linked lists

Start listNode\* rangeBetween (start, end)

{

a = 1;

b = 1;

Start listNode\* node1 = NULL\*, node2 = NULL;  
nodeb = NULL; nodea = NULL \* p & start;

until c = 0

while (ptr != NULL)

{

if (c == a - 1)

nodeb = ptr;

else if (c == a)

node1 = ptr;

else if (c == b + 1)

nodea = ptr;

break;

}

c + 1;

ptr = ptr-&gt;next;

Start listNode\* pre = nodea, \* temp;  
ptr = start;  
c = 0;

while (ptr != NULL)

{  
if (c >= a & & c < b)

temp = pto->next; initial  
~~pto->next = pto;~~  
 pto = pto->next; (initial) & (while loop)  
 pto = temp; (while loop)  
 } (while loop)  
 else if (c == b)  
 {  
 pto->next = pto->next->next; initial  
 if (a == 0)  
 {  
 start = pto->data + 100; initial  
 end = start + 100; initial  
 node((b)->next = pto); initial  
 break; initial  
 } (if b -> next & start == end)  
 else (start < end)  
 pto = pto->next;  
 if (\*c == \*j) (initial)  
 {  
 cout << start; initial  
 cout << endl; (after this)  
 cout << start->data; initial  
 } (if start == base)  
 i++; (initial)

Omtrent:

(c) header =  $T[1,2,3,4,5]$  will traverse  
~~left = fun = (first - next) + 1~~ smaller  
2      i + 1 <= first + next

$$\text{right} = y$$

anticipitib[us] (14, 3, 2, 5) ver. 6. part

Expected (1, 4, 3, 2, 5)

(-2)

head: { $\frac{1}{3}$ }  $\rightarrow$  students [ $\frac{1}{3}$ ]

left side of page 5 Exhibit 5

`right = 15 (11011 : 1 111)`

25/11/24 LAB-5 - Single Linked List C/C++ program  
 #include <stdio.h>  
 #include <stdlib.h>  
 struct Node  
 {  
 int data;  
 struct Node\* next;  
 };  
 struct Node\* createNode(int value)  
 {  
 struct Node\* newnode = (struct Node\*)  
 malloc(sizeof(struct Node));  
 newnode->data = value;  
 newnode->next = NULL;  
 return newnode;  
 }  
 void insertEnd(struct Node\*\* head, int value)  
 {  
 struct Node\* newNode = createNode(value);  
 if (\*head == NULL)  
 \*head = newNode;  
 else  
 {  
 struct Node\* temp = \*head;  
 while (temp->next != NULL)  
 temp = temp->next;  
 temp->next = newNode;  
 }
 }  
 void display(struct Node\* head)  
 {  
 struct Node\* temp = head;  
 while (temp != NULL)
 }

```
printf("%d", temp->data);
temp = temp->next;
}
printf("NULL");

void sortlinkedlist(Stenct Node* head)
{
    int swapped;
    Stenct Node* pto;
    Stenct Node* l;
    if(head == NULL)
        return;
    do
    {
        swapped = 0;
        pto = head;
        while(pto->next != l)
        {
            if(pto->data > pto->next->data)
            {
                int temp = pto->data;
                pto->data = pto->next->data;
                pto->next->data = temp;
                swapped = 1;
            }
            pto = pto->next;
        }
        l = pto;
    } while(swapped);
}

Stenct Node* reverislinkedlist(Stenct Node* head)
{
    Stenct Node* prev = NULL;
    Stenct Node* curr = head;
    while(curr != NULL)
    {
        curr->next = prev;
        prev = curr;
        curr = curr->next;
    }
    return prev;
}
```

Date:

current = next - prev;

```

printf("Sorted: ");
display(list1);
list2 = reverse(list2);
printf("Reversed 2");
display(list2);
concatenate(list1, list2);
printf("Concatenated list");
display(list1);
struct Node *temp;
while (list1 != NULL) {
    temp = list1;
    list1 = list1->next;
    free(temp);
}
return 0;
}

```

### Output:

```

Sorted list: 1 2 3 4 5 6 7 8
Enter no. of elements: 8
Enter elements: 1 2 3 4 5 6 7 8
Sorted list: 1 2 3 4 5 6 7 8
Enter the no. of element: 4
Enter the element: 4
Sorted list: 1 2 3 4 5 6 7 8
Rev -> 8 7 6 5 4 3 2 1

```

Concatenated:  $1 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 1 \rightarrow 1 \rightarrow 1$

Stack:-

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

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

```
struct Node {
```

```
int data;
```

```
struct Node* next;
```

```
};
```

```
struct Node* createNode(int value){
```

```
struct Node* newNode = (struct Node*)
```

```
malloc(sizeof(struct Node));
```

```
newNode->data = value;
```

```
newNode->next = NULL;
```

```
return newNode;
```

```
}
```

```
void push(struct Node** top, int value){
```

```
struct Node* newNode = createNode(value);
```

```
NewNode->next = *top;
```

```
*top = newNode;
```

```
}
```

```
int pop(struct Node** top) {
```

```
if (*top == NULL) {
```

```
printf("Underflow");
```

```
return -1;
```

```
}
```

```
struct Node* temp = *top;
```

```
int poppedValue = temp->data;
```

```
*top = temp->next;
```

```
free(temp);
```

```
return poppedValue;
```

```
}
```

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

```
int main() {
    struct Node* top = NULL;
    struct Node* newnode;
    int choice, value;
    do {
        printf("1. Push");
        printf("2. Pop");
        printf("3. Display");
        printf("4. Exit");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
```

```
                printf("Enter value: ");
                scanf("%d", &value);
                push(&top, value);
                break;
```

case 2:

```
    value = pop(&top);
    if (value == -1) {
        printf("Popped: %d\n", value);
    }
    break;
```

case 3:

```
    displayStack (top);
    break;
```

Date: / /

case 4: if \*stack is not empty  
printf("Existing "); // not "Valid"  
break;  
default: cout << "b" // printing  
printf(" Invalid"); // not valid

3  
while (choice != 4);  
Stack Node \*temp; // new tree  
while (top != NULL); // it prints  
temp = top; // prints tree  
top = top->next; // ab  
free (temp); // not "Valid"  
cout << endl; // not "Valid"  
3  
return 0; // (return 0) prints  
3

Output : ((initially "b" is present),  
(now) not "Valid")

Enter choice:

Enter value: 3 - value "Not valid"

Enter a "b" // prints

Stack OP: ((initially "b" is present),  
1. Push

2. Pop

3. Display

4. Exit

Enter your choice: 2 - value "Not valid"

Popped value: 3

Stack OP:

1. Push

2. Pop

3. Display  
4. Display  
Enter choice: 3  
stack 21

Queue:

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

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

struct Queue {
    struct Node *front;
    struct Node *rear;
};

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

struct Queue *createQueue() {
    struct Queue *queue = (struct Queue *) malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

void enqueue(Queue *queue, int value) {
    struct Node *newNode = createNode(value);
    if (queue->front == NULL) {
        queue->front = queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}
```

`struct Node * newNode = createNode(data);`  
 if (queue == NULL) {  
 queue->front = queue->rear = newNode;  
 return;
 }

`} queue->rear->next = newNode;`  
`queue->rear = newNode;`

`) int degene ( struct Queue * queue ) {`  
 if (queue->front == NULL) {  
 printf("Underflow!\n");
 } else {
 return;
 }

`} (x) * 1-1234567890`

`struct Node * temp = queue->front;`  
`int degeneratedValue = temp->data;`  
`queue->front = temp->next;`  
`if (queue->front == NULL) {`  
`queue->rear = NULL;`

~~(and it's time) stack overflow + stack underflow~~

~~(stack underflow)~~ free (temp); ~~\* stack underflow~~

~~(degenerated value)~~

~~int i = 0; i < 10; i++~~

~~r = displayQueue ( struct Queue \* queue ) {~~  
~~struct Node \* temp = queue->front;~~  
~~printf (" que ");~~

~~while (temp != NULL) {~~

~~printf("%d", temp->data);~~

~~temp = temp->next; i++;~~

~~printf("\n");~~

~~int main() {~~

~~struct Queue \* queue = createQueue();~~

```

unit choice & value;
do {
    printf("1. Insert & Enqueue");
    printf("2. Dequeue");
    printf("3. Display");
    printf("4. Exit");
    printf("Enter choice");
    scanf("%d", &choice);
    switch (choice) {

```

Case 1:

```

        printf("Enter value to enqueue");
        scanf("%d", &value);
        enqueue(queue, value);
        break;
    }

```

Case 2:

```

    value = degree(queue);
    if (value != -1) {
        printf("Deg value: %d", value);
        break;
    }

```

Case 3:

```

    displayQueue(queue);
    break;

```

Case 4:

```

    printf("Exiting");
    break;
    default:
        printf("Invalid");

```

while (choice != 0);

Struct node \*temp;

while (queue > front);

temp = queue[front];  
 queue[front] = queue[front + 1];  
 free(temp);  
 }  
 free(queue);  
 return;  
 }  
 cout << "Front : " << front;  
 cout << "Rear : " << rear;

Enter choice:

(Enter values to implement :-)

Queue op:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter choice: 2 ; "A" / Using

Dequeue value: 1

Queue op

1. Enqueue

2. Dequeue

3. Display

4. Exit

choice: 3

Queue : 2 3

Queue op

1. Enqueue

2. Dequeue

3. Display

4. Exit

choice : 4  
existing

✓  
~~25/1/24~~ 25/1/24 show Smith  
✓ 25/1/24 show Smith

✓ 25/1/24 show Smith  
✓ 25/1/24 show Smith

1/2/24

LAB-7

?

H: main.  
printing

```
# include <stdio.h>
# include <stdlib.h>
struct node {
    int data;
    struct node *prev;
    struct node *next;
};

struct node *s1=NULL;
struct node *createNode (int value)
{
    struct node *temp = (struct node *)
        malloc (sizeof (struct node));
    temp->data = value;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}
```

```
struct node *insert_left (struct node *
    start)
{
    int value, key;
    struct node *temp = createNode (0);
    printf ("Enter value to insert");
    scanf ("%d", &temp->data);
    printf ("Enter value to insert at node");
    scanf ("%d", &key);
    struct node *ptr = start;
    while (ptr != NULL && ptr->data != key)
```

```
, ptr = ptr->next;
```

```
if (ptr == NULL)
{
    printf("Node not found \n", key);
    free(temp);
}
else
{
    temp->next = ptr;
    temp->prev = ptr->prev;
    if (ptr->prev == NULL)
    {
        ptr->prev = temp;
    }
    else
    {
        ptr->prev->next = temp;
    }
    start = temp;
}
return start;
}

struct node * delete_value(struct
                           node * start)
{
    int value;
    printf("Enter value to delete: ");
    scanf("%d", &value);
    struct node * ptr = start;
    while (ptr != NULL && ptr->data == value)
    {
        ptr = ptr->next;
    }
    if (ptr == NULL):

```

```
printf("Node not found", value);
```

15

$\text{if}(\text{pt} \Rightarrow \text{prev} = \text{NULL})$

$\text{pto} \rightarrow \text{prev} \rightarrow \text{nex t} = \text{pto} \rightarrow \text{next}$

The given limit case

start $\leftarrow$  pto  $\rightarrow$  next $t[i]$   $\leftarrow$  s.pop

if ( p[0] == NULL )

$\text{pt} \rightarrow \text{hex} \rightarrow \text{prev} = \text{pt} \rightarrow \text{prev}$

```
printf("Node deleted");
```

frü (pto);

aktion statt;

~~and display construct mode \* start~~

Start node \* pto = start;

if (start == NULL)

```
printf("Empty");
```

else  
if

printf("list.");

while (ptr != NULL)

```
{  
    printf("%d\n", pto->data);  
    pto = pto->next;  
}  
}  
int main()  
{  
    int choice;  
    while(1)  
    {  
        printf("1. Create doubly linked list\n"  
              "2. Insert to left\n"  
              "3. Delete\n"  
              "4. Display contents\n"  
              "5. Exit\n");  
        scanf("%d", &choice);  
        switch(choice){  
            case 1:  
                s1 = createNode(0);  
                printf("Doubly linked list\n");  
                break;  
            case 2:  
                s1 = insertLeft(s1);  
                break;  
            case 3:  
                s1 = deleteValue(s1);  
                break;  
            case 4:  
                display(s1);  
                break;  
            case 5:  
                printf("Exiting\n");  
                exit(0);  
        }  
    }  
}
```

```
default:
    printf(" Invalid");
```

3

```
return;
```

3

antperf:

1. Create doubly linked list (1) ~~linked~~
2. Insert to left
3. Delete
4. Display contents
5. Exit

1

Doubly linked list created

1. Create doubly linked list (1) ~~linked~~
2. Insert to left (2) ~~linked~~ Repeat
3. Delete if head address (1) ~~linked~~ same for
4. Display contents (1) ~~linked~~ the rest
5. Exit

2

Enter value to insert: 11

Enter value to the left: 0

- 1.
- 2.
- 3.
- 4.
- 5.

2

Enter value to insert: 12

Enter value to left: 11

Sublist 1: 11  
1. 11 is not found in the list.  
2. Insert 11 to start of list.  
3. Insert 11 to end of list.  
4. Insert 11 to middle of list.

2. Insert 11 to start of list.

Enter value to insert: 13

Enter value to left: 12

1. 12 is not found in the list.  
2. Insert 13 to start of list.  
3. Insert 13 to end of list.  
4. Insert 13 to middle of list.

3. Enter value to delete: 12

Node with value 12 deleted.

1. Insert 13 to start of list.  
2. Insert 13 to end of list.  
3. Insert 13 to middle of list.

4. Insert 13 to start of list.  
5. Insert 13 to end of list.

List contents:

13 > 1 - normal form > (optional)

11

b : free form > (normal)

(normal) 11

5. Insert 13 to start of list.

leetcode - Split linked list in parts.

#include <stdlib.h>

struct ListNode\*\* splitListToParts (struct  
listNode\* head, int k, int \*  
returnSize)

2

struct listNode\* current = head;  
int length = 0;

while (current) {

length++;

current = current->next;

}

int part\_size = length / k;

int extra\_nodes = length % k;

struct listNode\*\* result = (struct listNode\*\*\*)  
malloc(k \* sizeof(struct listNode\*))

current = head;

for (int i=0; i<k; i++)

struct listNode\* part\_head = current;

int part\_length = part\_size + (i < extra\_nodes ? 1 : 0);

for (int j=0; j < part\_length - 1; j++) current = current->next;

current = current->next;

if (current)

struct listNode\* nextnode = current->next;

current  $\rightarrow$  next = NULL; : 8.3A selected  
 reset[i] = part\_head;  
 current = next\_node; starting #  
 } else { starting #  
 } won't work  
 i instead  
 reset[i] = NULL;  
 } don't work  
 }

aktion\_size \* k) not done?  
 k = 3 : action size for done  
 k = 2 } for (i = 0; i < k; i++) allow (\*stack + i)  
 then not

antpoint: stack - stack + stack with  
 stack < stack + stack - stack with  
 head = [1, 2, 3] with pointer  
 k = 5

first Antpoint is in \*stack not done  
 stack for 5000 \* stack not

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

(stack) shall start pointer

Expected

(stack) < door > stack in  
 stack [1, 2, 3, 4, 5, [ ]]

Red

(stack) < door > stack for 17/12/24

bus) shall start with giving door

(stack)

door numbers

## 15/2/24 LAB: 8 : Binary Search 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 = newNode->right = NULL;
    return newNode;
}

struct TreeNode* insertNode(struct TreeNode* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        root->right = insertNode(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);
```

```
}
```

```
void displayTree (struct TreeNode* root) {
```

```
printf("In-order:");
```

```
inOrder (root);
```

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

```
printf("Pre-order:");
```

```
preOrder (root);
```

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

```
printf("Post-order:");
```

```
postOrder (root);
```

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

```
int main () {
```

```
struct TreeNode* root = NULL;
```

```
int choice, data;
```

do 2

S (1000 + 1 from) 1.1.

```
printf("Insert"); answer choice in
printf("Display"); h & " If true
printf(" Exit"); answer choice
printf(" Enter your choice"); t
printf("%d", &choice); if bin
switch (choice) {
```

case 1: S (1000 + 1 from) 1.1.

```
: printf(" Enter"); h & " If true
:(+1) scanf("%d", &data);
:(+) root = insertNode (root, data);
break;
```

+ Case 2: answer choice in

(+) if (root == NULL) {

```
: printf(" Empty"); = 1 from
:(+3) else { answer choice in
:(+) if (choice == 1); answer choice
3 .(else) & root == "h" If true
break;
```

S (1000 + 1 from) 1.1. answer choice in
printf(" Exit"); h & " If true
break;
defeat; ("n/") If true
printf(" Invalid"); ("n") If true

} while (choice != 3); ("n") If true
}. answer choice in
("n") If true

? () now find
answer choice in
int b = 1;

output: Inorder traversal  
Data structure: BST

1. Insert node \* insert at root \* whole tree

2. Display

3. Exit { O = O(1) if (n = 1) } fn.

choice: 1

Data to insert: 10

1. Insert node \* start at root

2. Display { 1 = parent line }

3. Exit { 1 + new & down-right child }

choice: 1 & 2 = down-right

Data to insert: 40 { + N + parent }

1. Insert

2. Display

3. Exit

choice: 1

Data: 55

1. Insert

2. Display { n = 1 = 1 line } of

3. Exit { 1 = down-right }

choice: 1

1. Insert node \* whole tree

1. Insert

2. Display

3. Exit

choice: 2 { first (down-right) child }

In-order: 10 40 55 109

Pre-order: 10 40 55 109

Post-order: 109 55 40 10

choice: 1 { whole tree }

15/2/24

leet code - linked list

```

struct histNode* rotateRight(struct histNode* head, int k) {
    if (head == NULL || k == 0) {
        return head;
    }
    struct histNode* current = head;
    int length = 1;
    while (current->next != NULL) {
        current = current->next;
        length++;
    }
    int l = k % length;
    if (l == 0) {
        return head;
    }
    current = head;
    for (int i = 1; i < length - k; i++) {
        current = current->next;
    }
    struct histNode* newHead = current;
    current->next = NULL;
    current = newHead;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = head;
    return newHead;
}

```

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

k=2

antipnt [4, 5, 1, 2, 3] 26 show f[1]

Expected [4, 5, 1, 2, 3] 26 show f[2]

case: 2 case

[2, 0, 1, 2] case

26 tracing

26 show f[1]

26 show f[2]

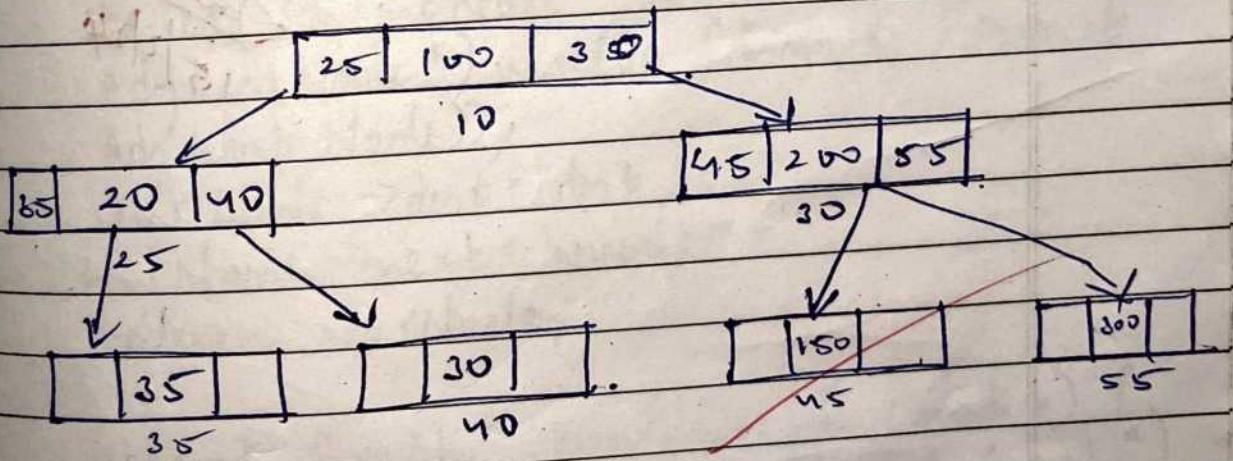
26 show f[3]

k=4

antipnt [2, 0, 1] 26 show f[1]

Expnt [2, 0, 1] 26 show f[2]

### TRACING of IAD -



Preorder

~~root1 = NULL; [S, N]~~

Point 100

left node 25 [S, S, N] Point 200

left node 35 Point 350

left node NULL [S, S, N] → price

Right node 30

Point 200

right node 45

Point 450

right left node 55

Point 300

left node NULL

→ rec

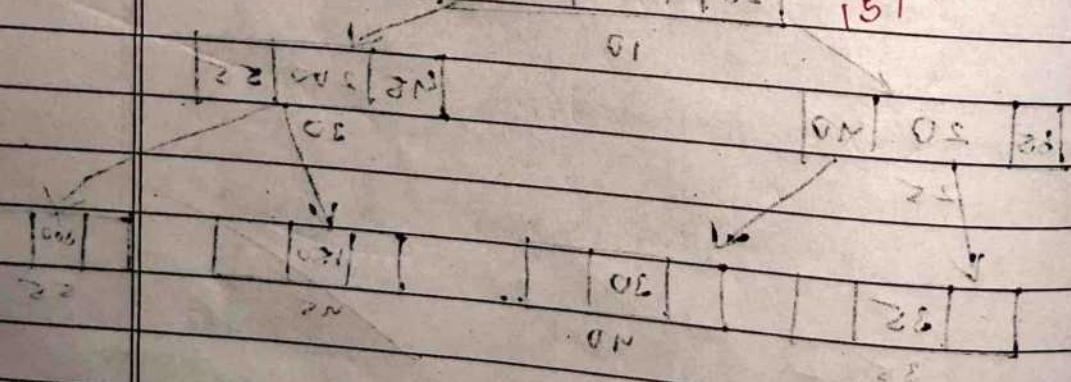
left node NULL right node NULL  
end.

→ go to PRACTICE

Next

10/14

10 20 30



Lab: 1  
DFS:BFS & DFS:

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

struct AdjlistNode {

int dest;

struct AdjlistNode \* next;

}; struct Adjlist {

struct AdjlistNode \* head;

}; struct Graph {

int V;

Adjlist \* array;

struct AdjlistNode \* newAdjlistNode (int dest) {

struct AdjlistNode \* newNode = (struct AdjlistNode \*) malloc (sizeof (struct AdjlistNode));

newNode-&gt;dest = dest;

newNode-&gt;next = NULL;

return newNode;

struct Graph \* createGraph (int v) {

struct Graph \* graph = (struct Graph \*)

malloc (sizeof (struct Graph));

graph-&gt;V = v;

graph-&gt;array = (struct Adjlist \*) malloc (v \* sizeof (struct Adjlist));

for (int i = 0; i &lt; v; i++)

graph-&gt;array [i].head = NULL;

return graph;

3

```
void addEdge (struct Graph* graph, int src, int dest) {
    struct AdjListNode* newNode = new
    AdjListNode (dest);
    newNode->next = graph->array [src];
    graph->array [src].head = newNode;
    newNode = new AdjListNode (src);
    newNode->next = graph->array [dest];
    head;
    graph->array [dest].head = newNode;
}
```

```
void DFSUtil (struct Graph* graph,
    int v, int* visited) {
    visited [v] = 1;
```

```
struct AdjListNode* temp = graph
    ->array [v].head;
```

```
while (temp != NULL) {
```

```
    int adjVertex = temp->dest;
    if (!visited [adjVertex])
```

```
        DFSUtil (graph, adjVertex, visited);
```

```
    temp = temp->next;
```

}

~~int isConnected (struct Graph\* graph,~~

~~int v) {~~

~~int \*visited = (int\*) malloc (v \* sizeof~~

~~int);~~

for (i=0; i<v; ++i)

```
visited[i] = 0;  
for(i=0; i<v; ++i)  
if(graph.h->array[i].head != NULL) {  
    DFSUtil(graph.h, i, visited);  
    break;  
}
```

```
y  
for(i=0; i<v; ++i)  
if(visited[i]==0)  
    return 0;  
    activate(i);  
y int main()  
{
```

```
int v, E;  
printf("Enter no. of vert");  
scanf("%d", &v);  
struct Graph* graph = createGraph(v);  
printf(" Edges");  
scanf(" %d", &E);  
printf(" Edges (%d)");  
for (int i=0; i<E; ++i)  
{
```

```
    int src, dest;  
    scanf(" %d %d", &src, &dest);  
    addEdge(graph, src, dest);
```

```
if(isConnected(graph, 1))  
    printf(" connected");  
else  
    printf(" Not connected");
```

X (Forward Link)

Output:

No. of Vertices: 5 for "out" option  
No. of Edges: 4 ; (v, "b, c") from

(i) Enter Edges: 0 to 1 edges pointing towards

0	1
1	2
2	3
3	4

Connected

Op/Py

BFS:

#include <stdio.h>

#include <stdlib.h>

struct AdjListNode {

int dest;

struct AdjListNode \*next;

struct AdjList;

struct AdjListNode \*head;

struct Graph {

int v;

struct AdjListNode \*\*array;

struct AdjListNode \*newAdjList  
Node (int dest) {

struct AdjListNode \*newNode = construct

(\*AdjListNode) malloc (sizeof (struct

(AdjListNode));

newNode->dest = dest;

NewNode->next = NULL;

return newNode;

struct Graph \*createGraph (int v)

struct Graph \*graph = (\*Graph \*) malloc (sizeof

(struct Graph));

graph->v = v;

graph->array = (\*AdjList \*) malloc

(v \* sizeof (struct AdjList));

for (int i=0; i<v; i++)  
graph->array[i].head = NULL;  
return graph;

y  
void addEdge (struct Graph\* graph,  
int src, int dest)  
{  
const AdjList\* newNode =  
new AdjList (dest);  
newNode->next = graph->array [src];  
graph->array [src].head = newNode;  
newNode = new AdjList (src);  
newNode->next = graph->array [dest];  
graph->array [dest].head = newNode;  
}

int isBFS (struct Graph\* graph, int  
start)  
{  
Node curr = graph->array [start];  
int \*visited = (int\*)malloc  
(graph->v \* sizeof (int));  
for (int i=0; i<graph->v; i++)  
visited [i] = 0;  
int \*queue = (int\*)malloc (graph  
->v \* sizeof (int));  
int front = 0, rear = 0;  
visited [i] = 1;  
int \*queue = (int\*)malloc (graph  
->v \* sizeof (int));  
while (front < rear) {  
int currentVertex = queue [front];  
}

int currentVertex = queue [front + 1];  
}

```
void printf(" %d", currentVertex);  
struct AdjListNode* temp; i=graph->  
array [ current Vertex ].head;  
while (temp) {  
    int adjVertex = temp->dest;  
    if (!visited [adj vertex]) {  
        visited [adj vertex] = 1;  
        queue [rear ++] = adj vertex;  
    }  
    temp = temp->next; i++;  
}  
free (visited);  
free (queue);
```

```
int main() {  
    int v, E;  
    printf(" Vertices ");  
    scanf("%d", &v);  
    printf(" Edges ");  
    scanf("%d", &E);  
}
```

```
struct Graph* graph = createGraph(v);  
printf(" Edges ");  
scanf("%d", &E);  
printf(" Enter edges ");  
for (int i = 0; i < E; i++) {  
    int src, dest;  
    scanf("%d %d", &src, &dest);  
    addEdge (graph, src, dest);  
}
```

```
int startVertex;  
printf(" Enter BFS ");  
scanf("%d", &startVertex);  
printf(" BFS, %d, startVertex);
```

BFS (graph, start vertex);  
action 0;

y

Output:

Enter vertices = 6

Enter edges = 7

Enter edges

0 1

0 2

1 2

1 3

2 3

3 4

4 5

start vertex = 0

traversal = 0 2 1 3 4 5

## 11/21/24 MACKER RANK

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

```
typedef struct Node {
```

```
    int data; // (int) size
```

```
    struct Node* left;
```

```
    struct Node* right; // (int) size
```

```
} 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;
```

```
    if (*index == 0) {
```

```
        Node* temp = root->left;
```

```
        root->left = root->right;
```

```
        root->right = temp;
```

```
}
```

```
    swapAtLevel(root->left, *index, *index + 1);
```

```
    swapAtLevel(root->right, *index, *index + 1);
```

```
}
```

```
int** swapNodes(int indexes[2],
```

mit index*i*'s children, init \* index  
 mit gemes count, init \* gemes, mit  
 result rows (init & result columns)  
 Node \* nodes + (Node \*<sup>2</sup>) nials ((  
     *N*. index is constant) \* size  
     (*N*. diff. (*N*. Node \*<sup>2</sup>))), it  
 for (unit i = 1; i <= index entries; i++)  
     nodes[i] = create Node(i);  
 }     | shoot down to following  
~~for (int i = 0; i <= 3; i++)~~  
     |     if (\* shoot down?  
antiparallel \* shoot down.  
         | shoot

| (upright down) shoot down \* shoot  
 | (downright down) shoot down \* shoot  
 | (downleft down) shoot down \* shoot  
 | (down) for (int i = 0; i <= 3; i++)  
     | - shoot = shoot < shoot down  
     | - i down = i down < shoot down  
     |     | down -> down -> shoot down  
     |     | shoot down < antiparallel  
     |

| - ; Expand antiparallel down  
 | (x down \* 3 + 1) -> shoot \* down  
 | (2, 1, 3) ; (down = down) for  
     | (0 == X) down for  
     | shoot < down = shoot \* shoot  
     | shoot < down = shoot < down  
     | shoot = shoot < down

| (1 + 2 \* 0, 3, 1 + 1) < down limit A down  
 | (1 + 2 \* 0, 1, 2, 1 + 1) < down limit B down

## LAB-10: Hashing at C-ii

```
#include <stdio.h>
#include <string.h>
#include <limits.h>
#define TABLE_SIZE 10
#define KEY_LENGTH 5
#define MAX_NAME_LENGTH 50
#define MAX_DESIGNATION_LENGTH 50
struct Employee {
    char key [KEY_LENGTH];
    char name [MAX_NAME_LENGTH];
    char designation [MAX_DESIGNATION_LENGTH];
    float salary;
};

struct HashTable {
    struct Employee *table [TABLE_SIZE];
};
```

```
int hashFunction(const char *key,
                  const struct HashTable *ht) {
    int sum = 0;
    for (int i = 0; key[i] != '\0'; i++) {
        sum += key[i];
    }
    return sum % ht->table_size;
}
```

```
int insert(struct HashTable *ht,
           const struct HashTable *ht,
           int index) {
    if (ht == NULL || ht->table == NULL) {
        return -1;
    }
    if (index < 0 || index > ht->table_size - 1) {
        return -1;
    }
    struct Employee *newEmployee =
        (struct Employee *) malloc(sizeof(struct Employee));
    if (newEmployee == NULL) {
        return -1;
    }
    newEmployee->key[0] = '\0';
    newEmployee->name[0] = '\0';
    newEmployee->designation[0] = '\0';
    newEmployee->salary = 0.0;
    ht->table[index] = newEmployee;
}
```

```
while (ht != NULL) {
    if (ht->table[index] == NULL) {
        index = (index + 1) % TABLE_SIZE;
    } else {
        printf("Employee at index %d: %s\n", index,
               ht->table[index]->name);
    }
}
```

~~ht -> table [index] = map;~~

struct Employee\* search (const  
HashTable& ht, const  
char\* key) {  
 int index = hashFunction(key);  
 if (ht.tableSize == 0)  
 return NULL;  
 else if (ht.tableSize == 1)  
 return ht.table[0];  
 else if (ht.tableSize > 1)  
 return ht.table[index];  
}

② While ( $ht \rightarrow \text{table}[index]$ )'s N-VL2) ;  
 if ( $s \in \text{comp}(\text{ht} \rightarrow \text{table}[index]) \rightarrow \text{key}$   
 [ $\text{HTN} - \text{N} + 1$  mod  $m$  key] == 0) {  
 [inserts  $s$  in  $ht \rightarrow \text{table}[index]$ ];  
 break; }  
 index = (index + 1) % tab[0].size;

```
    & std::vector<char> table; } line  
(const std::vector<Employee*>& empj  
char key [KEY_LENGTH];  
    if (!filter(file, & i)) {  
        char filename [100];  
        char line [100];  
        polint <= i < TABLE_SIZE; ++);  
        ht_table [i] = null;
```

~~Die Wahrheit besteht darin, dass die Jungs  
pointfile aufnehmen")~~

3. Learn of Christ ("walk in him")

3. Samo f. "v. s" (y almenos)

file = *poenit. tiliaceum*

if file = NUL {

$\text{avg } f(G) = N \vee L(L)$

~~(S-71-15107-7 E 5\*) D 11 21~~

```
while (ffgets (line, csize_of (line), file)) {
    emp = (struct Employee *) malloc (csize_of
        (struct Employee));
}
```

```
scanf ("%s.%s.%s.%f", emp->key, emp->name,
       emp->desig, emp->sal),
    insert (&ht, emp);
}
```

```
fclose (f6);
```

```
printf ("Key ");
```

```
scanf ("%s", key);
```

```
emp = search (&ht, key);
```

```
if (emp != NULL) {
}
```

```
printf (" %s ", emp, emp->key);
```

```
printf ("Name %s, ", emp->name);
```

```
printf ("Desig %s, ", emp->desig);
```

```
printf ("Sal %.2f\n", emp->sal);
```

```
}
```

```
else {
}
```

```
printf ("Not found %s, %s);
```

```
for (int i=0; i<TABLE_SIZE; i++) {
    if (ht.table[i] == NULL) {
        free (ht.table[i]);
    }
}
```

```
return 0;
}
```

Output:

Enter filename: employee.txt

key : 1223

Employee no 1223 : Name : Manager

Desig : CEO  
Salary : 1,000,000.00