INDEX

Name Ranisha Giri Std 3 Sec 3-D

Roll No. Subject DS School/College BMSCE

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

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
01	21/12/23	WAP for a) Push b) Pop c) Display		
02	28/12/23	Infix to Postfix conversion.		
03.	4/1/24	a) WAP to simulate working of a queue of integers. Provide: insertion, delete & display for queues. b) Circular queues.		
04.	11/1/24	WAP to implement singly linked list a) creation b) delete c) display) Leet code		
05	18/1/24	WAP to Implement SLL Leetcode.		
06.	25/1/24	Implement SLL :- Sorting, Reversed, Concatenation Stack Implementation & queues using S.L.L.		
07	1/2/24	Implement DLL :- Create, insert, delete & display Leet code		

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
08.	15/2/24	WAP a) Binary search tree b) traverse & c) display		
		Leetcode		
09.	22/2/24.	WAP to traverse a graph using BFS method.		
		WAP for DFS method.		
10.		Hashing linear probing		

Date 21 / 12 / 23
Page

3)
```
#include <stdio.h>
#include <conio.h>
void pop
#define N 10

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

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

void main()
{
   int ch;
   do
   {
      printf("Enter choice 1: Push \t 2: Pop \t 3: Display \n");
      printf("Enter your choice \n");
      scanf("%d", &ch);
      switch(ch){
         case 1: push();
                 break;
         case 2: pop();
                 break;
         case 3: display();
                 break;
         default: printf("Not valid value \n");
      }
   }while(ch!=0)
   getch();
}
```

```
void push()
{
    int x;
    printf("Enter your value\n");
    scanf("%d", &x);
    if (top == N)
    {
        printf("Overflow");
    }
    else {
        top++;
        stack[top] = x;
    }
void pop()
{
    int y;
    if(top == -1)
    {
        printf("Underflow");
    }
    else
    {
        y = stack[y];
        top--;
    }
void display()
{
    int i;
    for(i = top; i >= 0; i--)
    {
        printf("Elements are %d", stack[i]);
```

Date / /
Page

1 4
4

OUTPUT:-

Enter a choice
1: push 2: pop 3: display 4
Enter your choice
1
Enter a value
4

Enter a choice 1: push 2: pop 3: display 4
Enter your choice
2
~~Enter 0~~
4

Enter a choice 1: push 2: pop 3: display 4
Enter your choice
3

Ex 7
21/2/23

Date 28/12/23
Page

Lab 2:

1) Infix to Postfix conversion

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50

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

push (char elem)
{
    Stack [++top = elem;
}
char pop ()
{
    return (stack [top--]);
}
int pr (char symbol)
{
    if (symbol == '^')
    {
        return (3);
    }
    else if (symbol == '*' || symbol == '/')
    {
        return (2);
    }
    else if (symbol == '+' || symbol == '-')
    {
        return (1);
    }
    else {
```

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

Output:-

Enter the infix expression : A*B+C*D-E

Postfix expression : AB*CD*+E-

Date __/__/__
Page _____

Pgm 2:- Postfix Evaluation code

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

void push (int x)
{
    stack[++top] = x;
}
int pop ()
{
    return stack[--top];
}
int main ()
{
    char exp[20];
    char *e;
    int a,b,c,num;
    printf ("Enter the expressions :: ");
    scanf ("%s", exp);
    e = exp;
    while (*e != '\0')
    {
        if (isdigit (*e))
        {
            num = *e - 48;
            push (num);
        }
        else {
            a = pop();
            b = pop();
            switch (*e)
```

```
{
    case '+':
    {
        c = a + b;
        break;
    }
    case '-':
    {
        c = a - b;
        break;
    }
    case '*':
    {
        c = a * b;
        break;
    }
    case '/':
    {
        c = a / b;
        break;
    }
    }
    push(c);
}
e++;
}
printf("The result of expression is %s = %d",
        exp, pop());
}
```

Date __/__/__
Page _____

Output :-

Enter the expression : 12* 34* +5 -

The result of expression 12* 34* +5 = -9

Sg.f.

28/12/23

Lab 3 programs:

Queue:-

→
```
#include <stdio.h>
#define MAX 50
int queue_array[MAX];
int rear = -1;
int front = -1;
display ()
{
   int i;
   if (front == -1)
      printf ("Queue is empty \n");
   else
   {
      printf ("Queue is: \n");
      for (i = front; i <= rear; i++)
         printf ("%d", queue_array[i]);
      printf ("\n");
   }
}
void main ()
{
   int ch;
   while (1)
{
   printf ("1. Insert \n");
   printf ("2. Delete \n");
   printf ("3. Display \n");
   printf ("4. Exit \n");
   printf ("Enter your choice \n");
   scanf ("%d", &ch);
   switch (ch)
}
```

Date __/__/__
Page _____

```
case 1: insert ();
        break;
case 2: delete ();
        break;
case 3: display ();
        break;
case 4: exit (1);
        break;
default :
        printf ("Invalid \n");
}
}
}
void insert (int x)
{
   if (rear == MAX-1)
   {
      printf ("Overflow");
   }
   else if (rear == -1 && front == -1)
   {
      front = rear = 0;
      printf ("Insert element to queue :-");
      queue[rear] = x;        scanf ("%d", &x);
   }
   else
   {
      rear + +;
      queue[rear] = x;
   }
}
```

```
void delete()
{
    if (front == -1 && rear == -1)
    {
        printf ("Underflow");
    }
    else if (front == rear)
        front = rear = -1;
    else
    {
        printf ("Element to be deleted: %d", queue[front]);
        front++;
    }
}
```

Circular queue:-

```
#include <stdio.h>
#define MAX 50
int queue[MAX];
int front = -1;
int rear = -1;
void enqueue();
void dequeue();
void display();

void main()
{
    int ch;
    while(1)
    {
```

```
printf ("1. Enqueue \t");
printf ("2. Dequeue \t");
printf ("3. Display \t");
printf ("4. Exit \t");
printf ("Enter your choice \n");
scanf ("%d", &ch);
switch (ch)
{
    case 1: enqueue ();
            break;
    case 2: dequeue ();
            break;
    case 3: display ();
            break;
    case 4: exit (1);
            break;
    default: printf ("Invalid");
}
}
void enqueue (int x)
{
    if (front == -1 && rear == -1)
        printf ("Queue is full \n");
    else
    {
        if (front == -1)
        front = 0;
        rear = (rear + 1) % MAX;
        queue [rear] = x;
        printf ("inserted %d", x);
    }
}
```

```
void dequeue ()
{
  int ai
  if ~~(isEmpty ())~~ (front == -1 && rear == -1)
      printf ("Empty");
  else if (front == rear)
      {
          front = rear = -1;
      }
  else {
          printf ("%d", "Element to be deleted %d"
                  queue (front));
          front = (front + 1) %. MAX;
      }
}
void display ()-
{
    int i = front;
    if (front == -1 && rear == -1).
          printf ("Underflow");
    Else.
    {
          printf ("Queue is :");
          while (i) = rear).
          {
              printf ("%d", queue (i));
              i = (i + 1) %. MAX;
          }.
          printf ("%d", queue [rear]);
    }
}.
```

Date __/__/__
Page _____

OUTPUT:

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

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

Enter your choice : 2.
4 was removed from queue.

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice : 3
Queue is empty.

11/1/24

Lab 4 programs:-

Singly linked list

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

typedef struct node {
        int data;
        struct node* next;
    } node;
}
Node* head = NULL;

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

void main ()
{
    int ch;
    while (1)
    {
        printf ("1. Insert at beginning \n");
        printf ("2. Insert at end \n");
        printf ("3. Insert at position \n");
        printf ("4. display \n");
        printf ("5. Exit \n");
        printf ("Enter choice :");
        scanf ("%d", &ch);
        switch (ch)
        {
            case 1: push ();
                    break;
```

```
Case 2:
    append();
    break;
case 3:
    insert();
    break;
case 4: display();
        break;
default: printf("Existing");
}
}
}
void push()
{
    Node *temp = (Node*) malloc (sizeof(Node));
    int new_data;
    printf ("Enter data in 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:");
    temp -> data = new_data;
    temp -> next = NULL;
    if (head == NULL)
        {
            head = temp;
        }, return;
```

```
Node* temp1 = head;
while (temp -> next != NULL)
    {
        temp1 = temp1 -> next;
    }.
temp1 -> next = temp;
}.
void insert ()
{
Node * temp = (Node*) malloc (sizeof (Node));
int new_data, pos;
printf ("Enter data");
scanf ("%d", &new_data);
printf ("Enter position:");
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;
temp -> next = temp;
}
```

Date __/__/__
Page _____

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

OUTPUT:

Enter choice : 1
Enter data in new node : 0

1. Insert at beg
2. Insert at end
3. Insert at pos
4. Display
5. Exit

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

1. Insert at beg
2. Insert at end
3. Insert at pos
4. Display
5. Exit

Enter choice : 4
0 -> 2 -> 1 -> NULL

Date 11/01/24
Page

LEETCODE:-

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

MinStack* minStackCreate() {
    MinStack *st = (MinStack*) malloc(sizeof(MinStack));
    if (st == NULL)
    {
        printf("Memory allocation failed");
        exit(0);
    }
    st->size = 5;
    st->top = -1;
    st->s = (int*) malloc(st->size * sizeof(int));
    st->minstack = (int*) malloc(st->size * sizeof(int));
    if (st->s == NULL)
    {
        printf("Memory allocation failed");
        free(st->s);
        free(st->minstack);
        exit(0);
    }
    return st;
}
void minStackPush(MinStack* obj, int val) {
    if (obj->top == obj->size-1)
    {
        printf("Stack is overflow");
    }
```

Date __/__/__
Page _____

```
else{
    obj -> top++;
    obj -> s[obj -> top] = val;
    if (obj -> top == 0 || val < obj -> minstack[obj -> top-1])
    {
        obj -> minstack[obj -> top] = val;
    } else {
            obj -> minstack[obj -> top] = obj -> minstack
                                          [obj -> top -1];
        }
    }.
}
void MainStack pop (MinStack* obj){
    int value;
    if (obj -> top == -1)
    {
        Printf ("Underflow");
    }.
else
    {
    value = obj -> s[obj -> top];
    return value;
    }.
}.
```

OUTPUT:-

O/P: [null, null, null, null, -3, null, 0, -2]

I/P: ["MinStack", "push", "push", "push", "getMin", "pop", "top", "getMin"]
[[], [-2], [0], [-3], [], [], [], []].

Lab 6:-

Linked list code: Delete & Display.

```
#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 end 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 a node from specific position");
      printf("\n 6: EXIT");
```

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

switch (option)
{
case 1:
    create_ll(&start);
    printf("\n Linked list created");
    break;
case 2:
    display(start);
    break;
case 3:
    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 != 0);
return 0;
}
```

```
void create_ll (struct node ** start)
{
    struct node * new_node, * ptr;
    int num;
    printf ("Enter -1 to end \n");
    printf ("Enter the data : \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
{
    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 node which
        has to be deleted :");
scanf ("%d", &loc);
struct node *ptr = *start;
struct node* ptr1 = NULL;
```

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

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 a node from specific position
6: EXIT
Enter your option : 1
Enter -1 to end.
Enter the data :
1
Enter the data : 2
Enter the data : 3
Enter the data : 4
Enter the data : -1

LINKED LIST CREATED.
***** MAIN MENU *****
1: Create a list
2: Display
3: Delete from beg
4: Delete from end
5: Delete from specific position
6: EXIT
Enter your option : 2
Enter 1 2 3 4

***** MAIN MENU *****
1: Create a list
2: Display
3: Delete node from beg

Date __/__/__
Page _____

4: Delete from end
5: Delete from specific position
6: EXIT
Enter your option : 3 → Display : 2 3 4.

***** MAIN MENU *****
1: Create a list
2: Display
3: Delete (beg)
4: Delete end
5: Delete Specific position Display : 2 3.
6: Exit
Enter your option : 4.

***** MAIN MENU *****
1: Create a list
2: Display
3: delete from beg
4: delete from end
5: delete from specific position
6: Exit
Enter your choice : 5
"Enter the location of the node which has to be deleted : 1

Display :
2.

Sri
18/1/24

LEETCODE:-

Reversed linked list:-

```
struct list node* reverseBetween (struct listNode* head,
                                  int left, int right) {
    struct node listnode * l = head;
    struct listnode * r = head;
    int difference = right - left;
    if (left == right){
        return head;
    }
    for (int i = 0; i < left - 1; i ++) -
      {
        l = l -> next;
    }
for(int i = 0; i < right - 1; i ++)
  {
    r = r -> next;
  }
printf ("%d\n%d\n", l -> val, r -> val);
while (difference >= 0)
  {
    int temp = l -> val;
    l -> val = r -> val;
    r -> val = temp;
    l = l -> next;
    r = l;
    for (int i = 0; i < difference - 2; i ++)
    {
        r = r -> next;
    }
    difference -= 2;
}
return head;
```

Date / /
Page

WEEK 6 :-

1) Singly linked list :- sort, reverse & concatenation

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

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

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

void insert (int data, int position );
void delete (int position );
void display ();
void sort ();
void reverse ();
void concat (node ** head 1, node ** head 2);

int main ()
{
  int data, choice, pos;
  printf (" 1. Insert \n 2. Delete 3. Exit \n choice :")
  scanf (" %d", &choice);
  while (choice != 3)
  {
      if (choice == 1).
      {
```

```
printf ("Enter data and position : ");
scanf ("%d%d", &data, &pos);
insert (data, pos);
printf (" Count : %d\n", count);
}
else if (choice == 2)
{
  printf ("Enter position :")
  scanf ("%d", &pos);
  delete (pos);
  printf ("Count : %d\n", count);
}
display ();
printf ("Enter choice :");
scanf ("%d", &choice);
}
printf ("Original linked list :\n");
display ();
sort ();
printf ("sorted linked list :\n");
display ();
reverse ();
printf ("Reverse linked list :\n");
display ();
head1 = head;
head = NULL;
insert (3,0);
insert (4,1);
insert (1,2);
display ();
concat (&head1, &head);
head = head1;
```

```
printf ("concatenation with the above linked list
        given:\n");
return 0;
}
void sort ()
{
    int i, j, min_index;
    node *i_node = head, *j_node = head, *min_node;
    for (int i = 0; i < count-1; i++, i_node = i_node->
                                                next)
    {
        for (j_node -> data < i_node -> data)
        {
            min_index = j;
            min_node = j_node;
        }
    }
    if (min_index != i)
    {
        int Temp = i_node -> data;
        i_node -> data = temp;
    }
}
}
void reverse ()
{
    node * prev = NULL, * next = NULL;
    while (head != NULL)
    {
        next = head -> next
        head -> next = prev
        prev = head;
```

```
    head = next; }
head = prev; }

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

}}
```

OUTPUT:-

Original linked list :

2 1 4 3 5

Sorted linked list :

1 2 3 4 5

Reversed linked list :

5 4 3 2 1

3 4 1

Concatenation with the above linked list give :

5 4 3 2 1 3 4 1

STACK USING LINKED LIST:-

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

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

node * head = NULL;
int count = 0;

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

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

```
else if (choice == 2)
{
    printf ("Integer popped = %d\n", delete());
    printf ("Count : %d\n", count);
}
display();
printf ("Enter choice : ");
scanf ("%d", &choice);
}
return 0;
}
void insert (int data)
{
    node * new_node = (node*) malloc (sizeof (node));
    new_node -> data = data;
    new_node -> next = head;
    head = new_node;
    count ++;
    return;
}
void display ()
{
    node * temp = head;
    printf ("Stack : ");
    while (temp -> next != NULL)
    {
        printf ("%d", temp -> data);
        temp = temp -> next;
    }
printf ("%d", temp -> data);
printf ("\n");
}
```

OUTPUT:-

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

Date 1/01/24
Page

WEEK 7:-

Doubly linked list

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

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

node * head = NULL;
int count = 0;

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

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

```
        scanf("%d", &pos);
        delete(pos);
        printf("count: %d\n", count);
    }
    display();
    printf("Enter choice:");
    scanf("%d", &choice);
}
return 0;
}
void insert(int data, int position){
    if (position == 0){
        node* new_node = malloc(sizeof(node));
        new_node -> data = data;
        new_node -> next = head;
        new_node -> prev = NULL;
        if (head != NULL) head -> prev = new_node;
        head = new_node;
        count++;
        return;
    } else if (position == count){
        node* new_node = malloc(sizeof(node));
        new_node -> data = data;
        new_node -> next = NULL;
        node* temp = head;
        while (temp -> next != NULL)
            temp = temp -> next;
        temp -> next = new_node;
        new_node -> prev = temp;
        count++;
        return;
    } else if (position > count || position < 0){
```

```
printf ("Unable to insert at given position \n");
return;
} else {
    node *temp = head;
    for (int i = 0; i < position - 1; i++).
      temp = temp -> next;
    node* new node = malloc (sizeof (node));
    new_node -> data = data;
    new node -> next = temp -> next;
    new_node -> prev = new_node;
   temp -> next = new_node;
    count ++;
   return;
  }
};
void delete (int element) {
   int position = 0; node * temp = head;
   if (head == NULL) {
     printf (" Element does not exist in list ");
     return;
   };
 if (position == 0) {
    node * temp = head;
    temp = temp -> next;
    temp -> prev = NULL;
     free (head);
    head = temp;
    count --;
   return;
 } else if (position == count -1) {
   node* temp = head;
   for (int i = 1; i < count -1 ; i ++).
```

```
        temp = temp -> next;
        temp -> prev = NULL;
        free (head);
        head = temp;
        count --;
        return;
    } else if (position == count - 1) {
        node * temp = head;
    for (int i = 1; i < count - 1; i ++)
            temp = temp -> next;
        node * temp1 = temp -> next;
        temp -> next = NULL;
        free (temp1);
        count --;
        return;
    } else if (position > count || position < 0) {
        printf ("Unable to delete at position \n");
        return;
    } else {
        node * temp = head;
        for (int i = 0; i < position ; i ++)
            temp = temp -> next;
        temp -> next -> prev = temp -> prev;
        temp -> prev -> next = temp -> next;
        free (temp);
        count --;
        return;
    }
}
void display (){
    node * temp = head;
    printf ("Linked list :");
```

```
while (temp->next != NULL) {
        printf ("%d", temp->data);
        temp = temp->next;
    }
printf ("%d", temp->data);
printf ("\n");
}
```

OUTPUT:-

1. Insert
2. Delete
3. Exit

Choice : 1
Enter data & position : 2 0
Count : 1
Enter choice : 1
Enter data & position : 3 1
Count : 2.
Linked list : 2 3
Enter choice : 1
Enter data & position : 4 0
Count : 3.
Linked list : 4 2 3
Enter choice : 1
Enter data and position : 5 1
Count : 4.
Linked list : 4 5 2 3
Enter choice : 2
Enter element : 2
Count : 3.
Linked list : 4 5 3
Enter choice : 3
Exit

Date / /
Page

~~OUTPUT~~: LEETCODE :-

```
struct ListNode** splitListToParts(struct ListNode* head,
int k, int* returnSize) {
struct ListNode* temp = head, int n = 0;
for(; temp != NULL; temp = temp->next, n++);
struct ListNode** lists = (struct ListNode**) malloc
                          (k * sizeof(struct ListNode*));
for(int i = 0; i < k; i++) list[i] = NULL;
int earlier_lists = n%k; size = n/k;
int current = 0; bool list_over = false;
temp = head;
*returnSize = k;
for(int i = earlier_lists; i > 0; i--){
        struct ListNode* temp1 = temp;
        lists[current++] = temp;
        for(int j = 0; j < size; j++) temp1 = temp1->
        temp = temp1->next;
        temp->next = NULL;
}.
if(temp == NULL) return lists;
for(int i = 0; i < k - earlier_list; i++){
        struct ListNode* temp1 = temp;
        if(temp1 == NULL) break;
        for(int j = 0; j < size - 1; j++)
        temp1 = temp1->next;
        temp = temp1->next;
        temp1->next = NULL;
}
return lists;
}
```

Date 15/4/14
Page

WEEK 8: BINARY TREE.

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

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

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

int main ()
{
    int data;
    insert (&root, 8 );
    insert (&root, 3 );
    insert (&root, 1 );
    insert (&root, 6 );
    insert (&root, 4 );
    insert (&root, 7 );
    insert (&root, 10);
    insert (&root, 14);
    insert (&root, 13);
    printf ("1.Preorder 2.Inorder 3. Postorder 4. Exit :");
    printf ("Enter your choice :");
```

```
scanf("%d", &choice);
while (choice != 4) {
    if (choice == 1) {
        preorder(&root);
        printf("\n");
    } else if (choice == 2) {
        inorder(&root);
        printf("\n");
    } else if (choice == 3) {
        postorder(&root);
        printf("\n");
    }
    printf("Enter choice:");
    scanf("%d", &choice);
}
}
void insert(node **root, int data) {
    if (*root == NULL) {
        node *new_node = malloc(sizeof(node));
        new_node->data = data;
        new_node->right = NULL;
        new_node->left = NULL;
        *root = new_node;
        return;
    }
    if (data < (*root)->data) {
        insert(&((*root)->left), data);
    } else if (data > (*root)->data) {
        insert(&((*root)->right), data);
    }
    return;
}
```

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

OUTPUT:-

1. Preorder
2. Inorder
3. Postorder
4. Exit

Choice :1

8 3 1 6 4 7 10 14 13

Enter choice : 2

1 3 4 6 7 8 10 13 14

Enter choice : 3

1 4 7 6 3 13 14 10 8

Enter choice : 4

Exit

LEETCODE:-

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

Date 8/4/24
Page

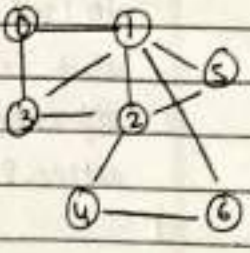
Week 9:

a) Traverse a graph using BFS method.

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

#define size 7

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

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

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

void bfs(int graph[][7]) {
    int visited[7];
```

```
for(int i=0;i<7;i++)
visited[i]=0;
push(0);
visited[0]=1;
while(fpos!=size){
  for(int i=0;i<7;i++){
    if(graph[queue[fpos]][i]==1 &&
        visited[i]!=1){
      push(i);
      visited[i]=1;
    }
  }
}
}
void push(int a){
  if(fpos==-1 && rpos==-1){
    queue[++rpos]=a;
    fpos++;
    return;
  }
else if(rpos==size-1){
    printf("Queue Underflow condition\n");
  }
int n=queue[fpos];
queue[fpos]=(int)NULL;
fpos++;
return n;
}
void display(){
  printf("Queue:");
  for(int i=0;i<size;i++)
```

BFS Traversal:-




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

Output:-

0 1 3 2 5 6 4 4

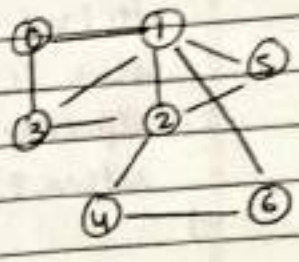
b) DFS method:-

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

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

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

int main(){
    int adj_matrix[7][7] = {
        {0,1,0,1,0,0,0},
        {1,0,1,1,0,1,1},
        {0,1,0,1,1,1,0},
        {1,1,1,0,0,0,1},
        {0,0,1,0,0,0,1},
        {0,1,1,0,0,0,0},
        {0,1,0,0,1,0,0},
    };
```

```
for(int i=0;i<7;i++)
visited[i]=0;
push(0);
visited[0]=1;
while(fpos!=rpos){
  for(int i=0;i<7;i++){
    if(graph[queue[fpos]][i]==1&&
        visited[i]!=1){
      push(i);
      visited[i]=1;
    }
  }
}
}
void push(int a){
  if(fpos==-1&&rpos==-1){
    queue[++rpos]=a;
    fpos++;
    return;
  }
else if(rpos==size-1){
    printf("Queue Underflow condition\n");
  }
int n=queue[fpos];
queue[fpos]=(int)NULL;
fpos++;
return n;
}
void display(){
  printf("Queue:");
  for(int i=0;i<size;i++)
```

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

BFS Traversal:-



Output:-

0 1 3 2 5 6 4

b) DFS method:-

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

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

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

int main(){
    int adj_matrix[7][7] = {
        {0,1,0,1,0,0,0},
        {1,0,1,1,0,1,1},
        {0,1,0,1,1,1,0},
        {1,1,1,0,0,0,1},
        {0,0,1,0,0,0,1},
        {0,1,1,0,0,0,0},
        {0,1,0,0,1,0,0},
    };
```

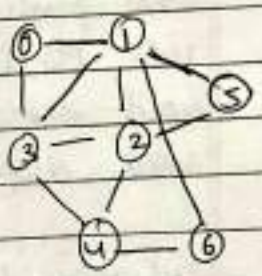
```
for (int i = 0; i < 7 ; i++)
stack [i] = NULL;
dfs (adj_matrix);
return 0;
}

void dfs (int graph [] [7]) {
  int visited [7];
  for (int i = 0; i < 7 ; i++)
  visited [i] = 0;
  while (pos != -1) {
    bool new_node = false;
    for (int i = 0; i < 7 ; i++) {
      if (graph [stack [pos]] [i] == 1 && visited [i] == 1);
        new_node = true;
        push (i);
        visited [i] = 1;
        printf ("%d", i);
        break;
      }
    }
  if (!new_node) pop ();
  }
}
void push (int a) {
  if (pos == size-1) {
    printf (" Stack Overflow condition");
    return;
  }
  stack [++pos] = a;
}
```

```
int pop(){
    if (pos == -1){
        printf ("stack Underflow condition");
        return (int) NULL;
    }
    return stack [pos -1];
}
void display (){
    for(int i=0; i < n; i++){
        printf ("%d", stack [i]);
    }
    printf ("\n");
}
```

Output :-

0 1 2 3 4 6 5

DFS Traversal :-



Hacker Rank Code :-

Sp:1

22/2/24

29/02/2024.

Lab 10 programs:

Hashing Using Linear Probing

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

#define size 10

int table [size];

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

int main () {
    for (int i = 0; i < size; i++)
    table [i] = -1;
    int choice;
    printf ("1. Insert \n 2. Delete \n 3. Display \n 4. Exit
                \n choice: ");
    scanf ("%d", &choice);
    int a;
    while (choice != 4) {
    switch (choice) {
        case 1:
            printf ("Enter integer to be pushed: ");
            scanf ("%d", &a);
            push(a);
            break;
```

```
case 2:
    printf ("Enter integer to be popped :");
        scanf ("%d", &a);
        int res = pop(a);
        if (res == 0)
        printf ("Integer popped \n");
        else
        printf ("Integer not found \n");
        break;
case 3:
    display ();
    break;
default:
        printf ("Idk");
        break;
    }
    printf (" Enter choice :");
    scanf ("%d", &choice);
}
}
void push (int data){
    int hash = data % size;
    while (table[hash] != -1 && hash <= (hash+size-1)%
            hash = (hash+1) % size;
    if (table[hash] == -1) table[hash] = data;
    else printf ("Table is full");
}
int pop (int data){
    int hash = data % size;
    for (int i =0; table[hash] != data ) || (i<size); i++, hash
            = (hash+1)% size);
        table[hash] = -1;
    return 0;
return -1;
```

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

OUTPUT:

1: Insert
2: Delete
3: Display
4: Exit
Choice: 1
Enter integer to be pushed: 12
choice: 1
Enter integer to be pushed: 23
choice: 1
Enter integer to be pushed: 45
choice: 1
Enter integer to be pushed: 5
choice: 1
Enter integer to be pushed: 78
choice: 2
Enter integer to be popped: 23
→ Integer popped
choice: 2
Enter integer to be popped: 78
Integer popped
Enter choice: 3
Table: -1 -1 12 -1 -1 45 5 -1 -1
Enter choice: 4
Exit

Date 14/4/24
Page

Lab 9 Programs:-

Hacker Rank:-

```
struct node {
    int data;
    struct node *left;
    struct node *right;
};
struct node * create_node (int val){
    if (val == -1) {
        return NULL;
}
void inorder ( struct node *root ) {
    if (!root){
        return;
    }
    inorder (root -> left);
    printf ("%d", root -> data);
    inorder (root -> right);
}
int max (int a, int b) {
    if (a>b){
        return a;
    }
    return b;
}
void enqueue ( struct node ** queue, struct node *root) {
        queue [tail] = root;
        tail ++;
}
int main() {
    int nodes_count, i, temp, h, tc_num, index, ide,
        temp1, temp2;
```

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

struct node *root_perm, *root_temp;
struct node *q[nodes_count];
  for (i=0; i<nodes_count; i++){
     q[i] = NULL;
}
while (tc.num){
  scanf ("%d", &inc);
  temp = inc;
  swap_nodes_at_level (root_perm, inc, 1, h);
  inorder (root_perm);
  printf ("\n");
  tc.num--;
}
return 0;
}
```

Output:-

Input:-

```
3
2 3
-1 -1
-1 -1
2
1
1
```

Output:-

```
3 1 2
2 1 3
```