INDEX

Name Nehal Std Sec

Roll No. Subject School/College

Street College Tel. No. 1BM22CS176 Parents Tel. No.

Sl. No.	Date	Title	Page No.	Teacher's Sign/ Remarks
1	7-12-23	Practice programs	10	
2	21-12-23	LAB 1	10	
3	28-12-23	LAB-2	10	
4	11-1-24	LAB - 3	10	
5	18-1-24	LAB - 4		
		LAB - 5		
6	25-1-24	LAB - 6	10	
7	1-2-24	LAB - 7	10	
8	15-2-24	LAB - 8	10	
9	22-2-24	LAB - 9		
10	29-2-24	LAB - 10		

Date __/__/__
Page _____

Week 1

Swapping 2 numbers using pointers

```
#include<stdio.h>
void swapping(int *x, int *y);
void main()
{
    int x,y;
    printf("Enter two numbers \n");
    scanf("%d %d", &x, &y);
    printf("The value of x and y before swapping
            is %d and %d \n", x,y);
    swapping(&x,&y);
    return 0;
    }
void swapping(int *x, int *y)
{
    int temp = *x;
    *x = *y
    *y = temp;
    printf("The value of x and y after swapping
            is %d and %d", *x, *y);
}
```

Output:

```
Enter two numbers
2
3
The value of x and y before swapping is 2 and 3
The value of x and y after swapping is 3 and 2
```

Write a program to stimulate the working of stack using an array with the following:

a) Push
b) Pop
c) Display

The program should print appropriate messages for stack overflow, stack underflow.

```
#include <stdio.h>
int stack[5];
int top=-1;
void push(int element);
void pop();
vid display();

int main(){
    int choice, elemt;
    do{
        printf("\n Stack Operations:\n");
        printf("1. Push \n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
      switch(choice)
      {
        case 1: printf("Enter element to push:");
                scanf("%d", &element);
                push(element);
                break;
        case 2: pop();
                break;
```

```
case 3:
    display();
    break;
case 4:
    printf("Exit program \n");
    break;
default:
    printf("Invalid choice \n");
}
} while (choice != 4);
return 0;
}

void push(int element) {
    if (top == 5 - 1) {
        printf("Stack overflow. Cannot push element.\n");
    } else {
        top++;
        stack[top] = element;
        printf("%d pushed to the stack.\n", element);
    }
}

void push(int element){
    if(top >= 5
void pop(){
 if(top == -1){
    printf("Stack underflow.\n");
  }
  else
  {
    printf("%d popped from the stack.\n", stack[top]);
    top--;
```

```
    }
}

void display()
  if (top == -1) {
     printf("Stack is empty. Nothing to display \n");
    }

    else
    {
    printf("Stack elements : ");
        for (int i=0; i <= top; i++) {
            printf("%d", stack[i]);
        }
        printf("\n");
    }
}
```

Output:

Stack operations:

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

Enter your choice : 1
Enter element to push : 4
4 pushed to the stack

Enter your choise : 1
Enter element to push : 6

6 pushed to stack

Enter your choice : 2
6 popped from the stack

Enter your choice : 3
Stack elements : 4

Enter your choice : 4
Exit program

3. WAP to implement dynamic memory allocation functions like malloc, free, calloc, realloc

```
#include<stdio.h>
int main()
{
int *ptr;
int n;
printf("Enter number of elements :");
scanf("%d", &n);
ptr = (int*) malloc(n * sizeof(int));
if(ptr == NULL)
{
printf("Memory not allocated.\n");
exit(0);
}
else {
printf("Memory successfully allocated using malloc.\n");
printf("Enter elements of array\n");
for(int i=0; i<n; i++){
scanf("%d", &ptr[i]);
}
```

```
printf("The elements of the array are : ");
for(int i = 0; i<n; i++) {
printf("%d", ptr[i]);
}
}
free(ptr);
printf("Memory Freed\n");
ptr = (int*) calloc(n, sizeof(int));
   if(ptr == NULL){
      printf("Memory not allocated.\n");
      exit(0);
   }
   else {
      printf("Memory successfully allocated using calloc.\n");
      printf("Enter elements of array\n");
      for(int i=0; i<n; i++) {
         scanf("%d", &ptr[i]);
      }
      printf("The elements of the array are : ");
      for(int i=0; i<n; i++)
      {
         printf("%d", &ptr[i]);
      }
      int n1;
      printf("Enter new size for realloc\n");
      scanf("%d", &n1);
      ptr = (int*) realloc(ptr, n1*sizeof(int));
      printf("Memory successfully re-allocated\n");
      printf("Enter more variables:\n");
      for(int i=n; i<n1; i++)
      {
         scanf("%d", &ptr[i]);
```

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

Output:

```
Enter number of elements: 4
Memory successfully allocated using malloc.
Enter elements of array
3
4
56
21
The elements of the array are: 3, 4, 56, 21, Memory Freed
Memory successfully allocated using calloc
Enter elements of array
3
4
32
7
The elements of the array are: 3, 4, 32, 7, Enter new size for realloc
8
Memory successfully re-allocated using realloc.
Enter more variables:
6
The elements of the array are: 3, 4, 32, 7,
```

Week 0

Output

1) Enter username : Nidal
Enter Account number: 1234
Enter deposit = 5000
Account created successfully

Enter amount to withdraw 2000
Account created successfully:
Balance amount = 7000

2) Enter number of string : 3
String 1 : Ball
String 2 : Tea
String 3 : Print
Sorted order: Ash coffee tea

3) Enter Array : 6 7 8 9 1 8 6 5 4 3 1
Enter key : 3
key 3 is found at position 9

4) Enter String : Hello
Enter substring : llo
Substring llo is found at position 3

5) Enter array : 4 5 3 1 6 3 8 8 7 4
Enter key : 8
Last occurence of key is found at 8 position

6) Enter array: 4 3 2 8 7 6 4
Enter key: 2
key 2 is found at 3 position

7) Enter array: 1 6 8 7 2 5 3 4
Enter key: 7
key 7 is found at position 4

8) Enter array: 5 35 6 7 81 20
Max element = 81
min element = 5

Week 2

Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply), / (divide) and ^ (power).

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

#define MAX_SIZE 100

int isOperator(char ch){
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/'
                || ch == '%');
}

int precedence (char operator){
    if(operator == '+' || operator == '-')
        return 1;
    if (operator == '*' || operator == '/' || operator == '%')
        return 2;
    return 0;
}

void infixToPostfix (char infix[], char postfix[]){
    char stack[MAX_SIZE];
    int top = -1;
    int i,j;

    for (i=0,j=0; infix[i] != '\0'; i++){
        if (infix[i] >= '0' && infix[i] <= '9'){
            postfix[j++] = infix[i];
```

```
}
else if (isOperator(infix[i]))
while(top>=0 && precedence(stack[top])>=precedence(infix[i]))
{
postfix[j++] = stack[top--];
}
stack[++top] = infix[i];
}
else if (infix[i] == '(')
{
stack[++top] = infix[i];
}
else if (infix[i] == ')')
{
while (top >= 0 && stack[top] != '(')
{
postfix[j++] = stack[top--];
}
if (top>=0 && stack[top] != '(')
top--;
}
}
}
while (top>=0)
{
postfix[j++] = stack[top--];
}
postfix[j] = '\0';
}
int main(){
char infix[MAX_SIZE], postfix[MAX_SIZE];

printf("Enter infix expression: ");
scanf("%s", infix);
```

```
    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);
    return 0;
}
```

Output : Enter infix expression : 2*8+3*6-5
Postfix expression : 28*36*5-

WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display. The program should print appropriate messages for queue empty and queue overflow conditions.

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

#define MAX_STACK_SIZE 100

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

void push(int item){
    if(top==MAX_STACK_SIZE-1)
    {
        printf("stack overflow\n");
        exit(EXIT_FAILURE);
    }

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

```
int pop(){
  if(top==-1){
    printf("Stack Underflow\n");
    exit(EXIT_FAILURE);
  }
  return stack[top--];
}

int isOperator(char ch){
  return(ch=='+'||ch=='-'||ch=='*'||ch=='/'
                          ||ch=='^');
}

int evaluatePostfix(char postfix[]){
int i=0;
while(postfix[i]!='\0'){
char currentSymbol=postfix[i];
if(isdigit(currentSymbol))
{
  push(currentSymbol-'0');
}
else if(isOperator(currentSymbol)){
  int operand2=pop();
  int operand1=pop();
  switch(currentSymbol){
  case '+':
    push(operand1+operand2);
    break;
  case '-':
    push(operand1-operand2);
```

```
            break;
        case '*':
            push(operand1 * operand2);
            break;
        case '/':
            push(operand1 / operand2);
            break;
        case '%':
            push(operand1 % operand2);
            break;
        }
    }
    i++;
}
return pop();
}
int main(){
    char postfixExpression[100];
    printf("Enter postfix expression : ");
    scanf("%s", postfixExpression);

    int result = evaluatePostfix(postfixExpression);
    printf("Result = %d\n", result);
    return 0;
}
```

Output: Enter postfix expression : 28*36*5-

Result : 13

Week 3

Date / /
Page

WAP to stimulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display. The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include <stdio.h>

#define max 6

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

void enqueue (int element)
{
    if (front== -1 && rear==-1)
    {
        front=0;
        rear=0;
        queue[rear]= element;
    }
    else if ((rear+1) % max == front)
    {
        printf ("Queue is overflow..");
    }
    else
    {
        rear=(rear+1) % max;
        queue [rear]= element;
    }
}
```

```
int dequeue()
{
  if((front==-1)&&(rear==-1))
  {
     printf("\n Queue is underflow..");
  }
else if(front==rear)
{
   printf("\n The dequeued element is %d",queue[front]);
   front=-1;
   rear=-1;
}
else
   {
      printf("\n The dequed elent is %d",queue[front]);
      front=(front+1)%max;
   }
}
void display()
{
   int i=front;
   if(front==-1 && rear==-1)
   {
      printf("\n Queue is empty..");
   }
   else
   {
      printf("\n Elements in a Queue are :");
      while(i<=rear)
      {
         printf("%d ",queue[i]);
         i=(i+1)%max;
      }
```

```
        {
    }
int main()
{
    int choice=1,x;
    while (choice<4 && choice!=0)
    {
        printf("\n Press 1: Insert an element");
        printf("\n Press 2: Delete an element");
        printf("\n Press 3: Display the element");
        printf("\n Enter your choice:");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1:
            printf("Enter the element which is to be inserted");
            scanf("%d",&x);
            enqueue(x);
            break;
            case 2:
            dequeue();
            break;
            case 3:
            display();
        }
    }
    return 0;
}
```

Output:

Enter 1 to insert, 2 to delete, 3 to display, 4 to exit:

Press 1 : Insert an element

Press 2 : Delete an element

Press 3 : Display the element

Enter your choice

1

Enter the element which is to be inserted

20

Element 20 inserted successfully.

Enter 1 to insert, 2 to delete, 3 to display, 4 to exit: 1

Enter the element to insert: 40

Element 40 inserted successfully.

Press 1: to insert an element

Press 2 : Delete an element

Press 3 : Display the element

Enter your choice

2

Deleted element: 20

Press 1 : to insert an element

Press 2 : Delete an element

Press 3 : Display the element

Enter your choice

3

40

Press 1 to insert an element

Press 2 : to delete an element

Press 3 : to display an element

Press 4 : Exit

Exiting the box

Week 4

Date / /
Page

4. WAP to implement Singly Linked List with following operations:

a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list.

c) Display the contents of the linked list

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

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

void push(struct Node ** head_ref, int new_data);
void append(struct Node ** head_ref, int new_data);
void insert_at_pos(struct Node ** head_ref, int position,
                   int new_data);
void display(struct Node * head);

int main() {
    struct Node * head = NULL;
    int choice, data, position;
    do {
        printf("\n Linked List Operations: \n");
        printf("1 Insert at the beginning \n");
        printf("2 Insert at the end \n");
        printf("3 Insert at a specific position \n");
        printf("4. Display the Linked list \n");
        printf("5. Exit \n");
```

```
    }
    struct Node *temp = *head_ref;
    printf("Node inserted at position %d.\n", position);
    return;
}

struct Node *temp = *head_ref;
for (int i=1; i<position-1 && temp!=NULL; i++)
temp = temp->next;
}
if (temp == NULL){
printf("Invalid position. Node not inserted.\n");
}
else
{
new_node->next = temp->next;
temp->next = new_node;
printf("Node inserted at position %d.\n", position);
}
}
void display (struct Node *head){
  printf("Linked List: ");
  while (head!=NULL){
    printf("%d->", head->data);
      head = head->next;
    }
    printf("NULL\n");
}
```

Output:

0. Create
1. Display
2. Insert Node at beginning
3. Insert Node in specific position
4. Insert Node at end of linked list
5. Delete Node at beginning
6. Delete Node at end
7. Delete node at position

Enter your choice : 0
Enter node data : 60

0. create

Enter your choice : 2
Enter node data : 70

0. create
Enter your choice : 2
Enter node data : 70

0. create
Enter your choice : 3
Enter node data : 80
Enter position : 1

0. create
Enter your choice : 4
Enter node data : 30

0. create
Enter your choice : 5
Node Deleted

0. create
Enter your choice : 1
Linked List : 80 60 30

0. create
Enter your choice : 8

Date __/__/__
Page _____

1. ~~Write a program to implement Singly Linked List with following operations~~
   a) ~~Create a linked list.~~
   b) ~~Deletion of first element, specified element and last element in the list.~~
   c) ~~Display the contents of the linked list~~

2. Demonstration of account creation on LeetCode platform and stack program given in below link.

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

typedef struct {
    int val;
    int min;
} Node;

typedef struct {
    Node * stack;
    int top, capacity;
} MinStack;

MinStack * minStackCreate () {
    MinStack * stack = (MinStack *) malloc (sizeof (MinStack
    stack -> stack = (Node *) malloc (sizeof (Node) * 100);
    stack -> top = -1
    stack -> capacity = 100;
    return stack;
}
```

Date __/__/____
Page _____

```
void minStackPush(MinStack* obj, int val) {
    if (obj->top == -1) {
        obj->stack[++(obj->top)].val = val;
        obj->stack[obj->top].min = val;
    }
    else
    {
        obj->stack[++(obj->top)].val = val;
        obj->stack[obj->top].min = (val < obj->stack
        [obj->top-1].min) ? val : obj->stack[obj->top-1].min;
    }
}

void minStackPop(MinStack* obj) {
    if (obj->top >= 0) {
        obj->top--;
    }
}

int minStackTop(MinStack* obj)
{
    return obj->stack[obj->top].val;
}

int minStackGetMin(MinStack* obj) {
    return obj->stack[obj->top].min;
}

void minStackPop(MinStack* obj) {
    if (obj->top >= 0) {
        obj->top--;
    }
}
```

Date __/__/__
Page _____

```
void minStackPush(MinStack* obj, int val) {
    if (obj->top == -1) {
        obj->stack[++(obj->top)].val = val;
        obj->stack[obj->top].min = val;
    }
    else
    {
        obj->stack[++(obj->top)].val = val;
        obj->stack[obj->top].min = (val < obj->stack
        [obj->top-1].min) ? val : obj->stack[obj->top-1].min;
    }
}

void minStackPop(MinStack* obj) {
    if (obj->top >= 0) {
        obj->top--;
    }
}

int minStackTop(MinStack* obj)
{
    return obj->stack[obj->top].val;
}

int minStackGetMin(MinStack* obj) {
    return obj->stack[obj->top].min;
}

void minStackPop(MinStack* obj) {
    if (obj->top >= 0) {
        obj->top--;
    }
}
```

```
int minStackTop (MinStack* obj) {
    return obj->stack[obj->top].val;
}

int minStackGetMin (MinStack* obj) {
    return obj->stack[obj->top].min;
}

void minStackFree (MinStack* obj) {
    free(obj->stack);
    free(obj);
}
```

Output:

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

18/1/24

Date __/__/____
Page _____

Demonstration of LeetCode program on Singly Linked List. Reverse Linked List.

```
struct ListNode* reverseBetween(struct ListNode* start, int a,
                                int b)
{
    a--;
    b--;
    struct ListNode *node1= NULL, *node2 = NULL, *nodeb=NU
          *nodea = NULL, *ptr=start;
    int c=0;
    while (ptr!=NULL)
    {
        if (c==a-1)
            nodeb=ptr;
        else if (c==a)
            node1=ptr;
        else if (c==b)
            node2=ptr;
        else if (c==b+1)
            {
                nodea=ptr;
                break;
            }
        c+=1;
        ptr=ptr->next;
    }
    struct ListNode *pre=nodea, *temp;
    ptr=start;
    c=0;
```

```
while (ptr != NULL)
{
    if(c>=a && c<b)
    {
        temp = ptr -> next;
        ptr -> next = pre;
        pre = ptr;
        ptr = temp;
    }
    else if(c==b)
    {
        ptr -> next = pre;
        if(a==0)
            start = ptr;
        else
            nodeb -> next = ptr;
        break;
    }
    else
        ptr = ptr -> next;
        c += 1;
    }
    return start;
}
```

Output

[1, 4, 3, 2, 5]

Week 6

Date __/__/__
Page _____

1. WAP to implement Single Linked List with following operations : Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *s1=NULL;
struct node *s2=NULL;
struct node *start=NULL;
struct node *create(struct node*);
void sort();
struct node *concatenate(struct node*, struct node*);
void reverse();
void display(struct node*);

int main()
{
    int option;
    struct node *a=NULL;
    do {
    printf("\n 1. Create Linked List \n 2. Create two Linked list for concat
            \n3. Sort \n 4. Concatenate \n5. Reverse \n6. Display
            \n 7. Display concat Linked List \n. 8 Exit \n");
    scanf("%d", &option)
    switch (option)
    {
        case 1: start= create(start);
```

```
        printf("\n Linked List created \n");
        break;
    case 2: printf("\n Linked List 1:\n");
            s1 = create(s1);
            printf("\n Linked List 2:\n");
            s2 = create(s2);
            printf("\n Linked List created\n");
            break;
    case 3: sort();
            printf("\n Linked List sorted\n");
            break;
    case 4: a = concatenate(s1, s2);
            printf("\n Linked list concatenated\n");
            break;
    case 5: reverse();
            printf("\n Elements list reversed\n");
            break;
    case 6: printf("Elements in the Linked list\n");
            display(start);
            break;
    case 7: printf("\n Elements after concatenation:\n");
            display(a);
            break;
        }
    } while(option!=8);
    return 0;
}

struct node * create(struct node *start)
{
    struct Node *ptr, *new_node;
    int num;
    printf("Enter -1 to exit\n");
```

```
printf ("\n Enter the data : ");
scanf ("%d", &num);
while (num != -1)
{
    new_node = (struct node *) malloc (sizeof (struct node));
    new_node -> data = num;
    if (start == NULL)
    {
        start = new_node;
        new_node -> next = NULL;
    }
    else
    {
        ptr = start;
        while (ptr -> next != NULL)
        ptr = ptr -> next;
        ptr -> next = new_node;
        new_node -> next = NULL;
    }
    printf ("Enter the data :");
    scanf ("%d", &num);
}
    return start;
}

void sort()
{
    struct node *i, *j;
    int temp;
    for (i = start; i -> next != NULL; i = i -> next)
    {
        for (j = i -> next ; j != NULL ; j = j -> next)
        {
```

```
        if (i->data > j->data)
        {
            temp = i->data;
            i->data = j->data;
            j->data = temp;
        }
    }
}

struct node *concatenate(struct node *t1, struct node *t2)
{
    struct node *ptr;
    ptr = t1;
    while (ptr->next != NULL)
    {
        ptr = ptr->next;
    }
    ptr->next = t2;
    return t1;
}

void reverse()
{
    struct node *prev = NULL;
    struct node *next = NULL;
    struct node *cur = start;
    while (cur != NULL)
    {
        next = cur->next;
        cur->next = prev;
        prev = cur;
        cur = next;
```

Date __/__/__
Page _____

```
        }
        start=prev;
}

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

2. WAP to Implement Single Linked List to simulate Stack

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *start=NULL;
void push();
void pop();
void display();
int main()
{
    int val, option;
```

```
    do
    {
        printf("\n Enter the number to perform
            \n1. Push \n2. Pop \n3. Display \n4. Exit
        scanf("%d", &option);
        switch(option)
        {
            case 1: push();
            break;
            case 2: pop();
            break;
            case 3: display();
            break;
        }
    } while(option != 4);
    return 0;
}
void push()
{
    struct node *new_node;
    int num;
    printf("Enter the data \n");
    scanf("%d", &num);
    new_nod = (struct node *) malloc(sizeof(struct node));
    new_node -> data = num;
    new_node -> next = start;
    start = new_node;
}
void pop()
{
    struct node *ptr;
    ptr = start;
    while(ptr != NULL)
```

```
    if (start = NULL)
    {
        printf("Stack is empty \n");
        exit(0);
    }
    else
    {
        ptr = start;
        start = ptr -> next;
        printf("\n Element pupped is %d \n", ptr -> data);
        free(ptr);
    }
}
void display()
{
    struct node *ptr;
    ptr = start;
    while (ptr != NULL)
    {
        printf("\t %d", ptr -> data);
        ptr = ptr -> next;
    }
    printf("\n");
}
```

Output

Enter the number to perform following operations

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

1.

Enter the data

Date __/__/__
Page _____

5

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

1.

Enter data

6

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

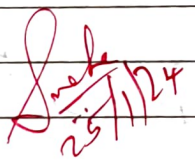
2.

Element popped = 6

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

3

5



23/11/24

Date __/__/__
Page _____

2. Queue implementation

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node* next;
};
struct node *start=NULL;
void enqueue();
void dequeue();
void display();
int main()
{
    int val, option;
    do
    {
        printf("\n Enter number \n 1. Enqueue \n 2. Dequeue
                \n 3. Display \n 4. Exit");
        scanf("%d", &option);
        switch(option)
        {
          case 1: enqueue();
          break;
          case 2: dequeue();
          break;
          Case 3: display();
          break;
        }
  }
  While (option!=4);
  return 0;
```

```
}
Void enqueue()
{
    struct node *new_node;
    int num;
    printf("Enter data \n");
    scanf("%d", &num);
    new_node = (struct node*) malloc(sizeof(struct node));
    new_node -> data=num;
    new_node -> next=start;
    start = new_node;
}
void dequeue()
{
   struct node *ptr, *preptr;
   ptr=start;
   if(start == NULL)
   {
      printf("Stack is empty\n");
      exit(0);
   }
   else if(start -> next==NULL)
   {
      start= start -> next;
      printf("\n Element from the stack is: %d \n", ptr->data);
      free(ptr)
   }
   else
   {
     while(ptr -> next != NULL)
     {
     preptr -> next=NULL;
     printf("\n Element popped from stack is %d \n", ptr->data);
```

```
        free(ptr);
        }
}
Void display()
{
    struct node *ptr;
    ptr=first;
    while (ptr != NULL)
    {
        printf ("\t %d", ptr->data);
        ptr = ptr->next;
    }
}
```

Output:

Enter the number to perform
1. Enqueue
2. Dequeue
3. Display
4. Exit
1
Enter the data
10
Enter the number to perform following operations.
1. Enqueue
2. Dequeue
3. Display
2
Element popped from stack is 10

1-2-24

Date __/__/__
Page _____

LAB - 7

WAP to Implement doubly link list with primitive operations.

a) Create a doubly linked list
b) Insert a new node to the left of the node
c) Delete the node on a specific value

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *prev;
    struct node *next;
};
struct node *sl = NULL;
struct node *creatNode (int vale){
    struct node *temp=(struct node*) malloc (sizeof(struct node));
    temp->data= vale;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}
struct node *insert_left (struct node *head) {
    int value, key;
    struct node *temp= creatNode(0);
    printf("Enter vale");
    scanf("%d", temp->data);
    printf("Enter value to the left of which the node is to
        be inserted");
    scanf("%d", &key);
    struct node *ptr= head;
```

```
while ( ptr != NULL && ptr -> data != key)
{
    ptr = ptr -> next;
}
if (ptr = NULL)
{
    printf ("No node with value %d", &key);
    free (temp);
}
else
{
    temp -> next = ptr;
    temp -> prev = ptr -> prev;
    if (ptr -> prev = NULL)
    {
        ptr -> prev -> next = temp;
    }
    ptr -> prev = temp;
    if (ptr == head) {
        head = temp;
    }
}
return head;
}
struct node *delete_value (struct node *start) {
    int value;
    printf ("Enter value to be deleted");
    scanf ("%d", &value);
    struct node * ptr = head;
    while (ptr != NULL && ptr -> data != value)
    {
        ptr = ptr -> next;
    }
```

```
    if (ptr = NULL)
    {
        printf(" %d value does not exist ", value);
        free(ptr);
    }
    else
    {
        if(ptr->prev != NULL)
        {
            ptr->prev->next = ptr->next;
        }
        else
        {
            head = ptr->next;
        }
        if( ptr->next != NULL)
        {
            ptr->next->prev = ptr->prev;
        }
        printf("Node with value %d", value);
        free(ptr);
    }
    return head;
}
void display (struct node *head)
{
    struct node *ptr = head;
    if (ptr == NULL)
    {
        printf("List is empty \n");
    }
    else
    {
```

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

struct node * insert_right(struct node *head)
{
    int value;
    struct node *temp = createNode(0);
    printf("Enter value to inserted");
    scanf("%d", &temp->data);
    if (head == NULL)
    {
        head = temp;
    }
    else
    {
        struct node *ptr = head;
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        ptr->next = temp;
        temp->prev = ptr;
    }
    return start;
}
```

```
int main(){
  int choice;
  while(1){
  printf("1.Create a doubly linked list \n 2. Insert to left
       \n 3. Delete based on value \n 4. Display");
  scanf("%d", &choice);
    switch(choice){
      case 1:
        sl = createNode(0);
        printf("Doubly linked list created");
      break;
      case 2:
        sl = insert_left(sl);
        break;
      case 3:
        sl = deletevalue(sl);
      case 4:
        display(sl);
        break;
      case 6:
        printf("Exit\n");
        exit(0);
      default:
        printf("Invalid choice \n");
    }
  }
  return 0;
}
```

Date __/__/__
Page _____

Output:

1. Create doubly linked list
2. Insert to the left
3. Delete based on value.
4. Display contents
5. Exit

1.

Created

4.

Enter value to insert: 12

4

Enter value to insert 6

2.

Enter value to the left of the node to be inserted: 3

4.

List content:

12

3

6

Smoke
1/2/24

Date / /
Page

8. Leet code program

```
#include <stdlib.h>
struct ListNode** splitListToParts(struct ListNode* head,
  int k, int *returnSize){
    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 && current; j++)
       {
          current = current->next
       }
       if (current){
          struct ListNode* next_node= current->next;
          current->next = NULL;
          result[i]= part_head;
          current = next_node;
          }
          else {
             result[i] = NULL;
```

```
        }
    }
    *returnSize = k;
    return result;
}
```

Output

case 1

head = [1, 2, 3]

k = 5

Output = [[1], [2], [3], [], []]

case 2: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

k = 3

Output = [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]]

15-2-24

Date __/__/____
Page _____

Week 8

1. Write a program to construct a binary search tree, to traverse the tree using all the methods i.e, in-order, preorder and postorder To display the 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= NewNode->right=NULL;
    return newNode;
}
struct Tree* insertNode ( struct Tree* 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 inOrder (struct TreeNode *root)
{
    if (root != NULL)
    {
        inOrder (root->left);
        printf ("%d ", root->data);
        inOrder (root->right);
    }
}
void preOrder (struct TreeNode *root){
    if (root != NULL){
        printf ("%d ", root->data);
        preOrder (root->left);
        preOrder (root->right);
    }
}
void postOrder (struct TreeNode *root)
{
    if (root != NULL)
    {
        postOrder (root->left);
        postOrder (root->right);
        printf ("%d ", root->data);
    }
}
void displayTree (struct Tree *root){
    printf (" In order : ");
    inorder (root);
    printf (" Pre-order : ");
    preOrder (root);
```

```
        printf("Post-order: ");
        postOrder(root);
}

int main(){
    struct TreeNode* root = NULL;
    int choice, data;
    do {
        printf("1. Insert a node \n");
        printf("2. Display tree \n");
        printf("3. Exit");
        scanf("%d", &choice);
        switch(choice)
        {
        case 1:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            root = insertNode(root, data);
            break;
        case 2:
            if(root == NULL)
            {
                printf("Tree is empty \n");
            }
            else
            {
                displayTree(root);
            }
            break;
        case 3:
            printf("Exiting program \n");
            break;
        default:
            printf("Invalid choice \n");
```

Date __/__/__
Page _____

```
} while (choice != 3);
    return 0;
}
```

Output:

```
1. Insert a node
2. Display tree
3. Exit
Enter choice : 1
Enter data : 50
1. Insert a node
2. Display tree
3. Exit
Enter choice : 1
Enter data : 20
1. Insert a node
2. Display tree
3. Exit
Enter choice : 1
Enter data : 70
2. Display tree
In-order traversal : 20 50 70 80
Pre-order traversal : 50 20 70 80
Post order traversal : 20 80 70 50
4. Exit
```

Date __/__/__
Page

2. leet code- Linked list

```
struct ListNode* rotateRight(struct ListNode *head, int k){

    if(head==NULL || k==0){
        return head;
    }

    struct ListNode *ptr=head;
    int length=1;
    while(ptr->next!=NULL)
    {
        ptr=ptr->next;
        length++;
    }

    k=k%length;
    if(k==0)
    {
        return head;
    }
    ptr=head;
    for(int i=1; i<length-k; i++){
    {
        ptr=ptr->next;
        ptr=newhead;
    while(ptr->next!=NULL)
    {
        ptr=ptr->next;
    }
    ptr->next=head;
    return newHead;
}
```

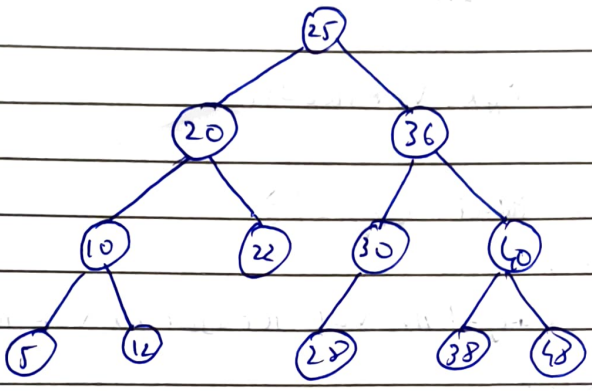
Output :

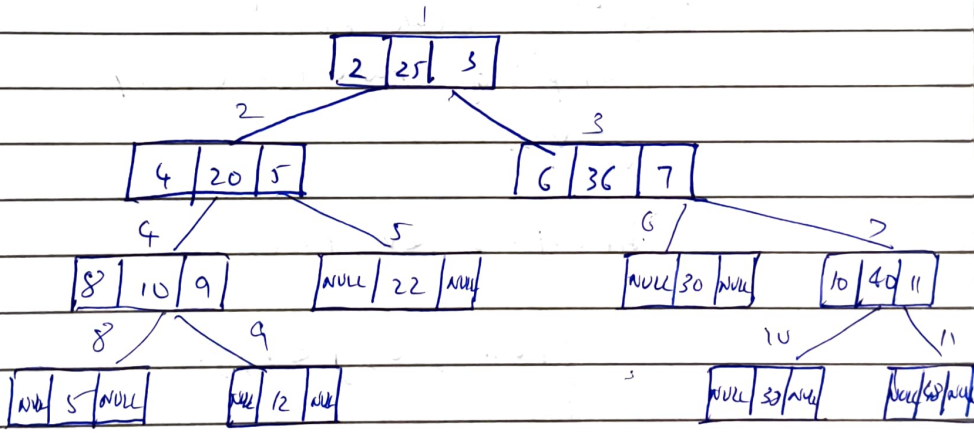
Input : head = [1, 2, 3, 4, 5]

k = 2

Output : [4, 5, 1, 2, 3]

Tracing





inorder - 5 10 12 20 22 25 28 30 36 38 40 48

Preorder - 25 20 10 5 12 22 36 30 28 38

Postorder - 5 12 10 22 20 28 30 48 38 36 25

Date
Page

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

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

#define MAX_VERTICES 50

typedef struct Graph_t {
    int V;
    bool adj[MAX_VERTICES][MAX_VERTICES];
} Graph;

Graph *Graph_create(int V) {
    Graph *g = malloc(sizeof(Graph));
    g->V = V;

    for (int i=0; i<V; i++)
    {
        for (int j=0; j<V; j++)
        {
            g->adj[i][j] = false;
        }
    }
    return g;
}
void Graph_destroy(Graph *g)
{
    free(g);
```

```
}
void Graph_addEdge(Graph *g, int v, int w)
{
    g->adj[v][w] = true;
}

void Graph_BFS(Graph *g, int s)
{
    bool visited[MAX_VERTICES];
    for(int i=0; i<g->V; i++)
    {
        visited[i] = false;
    }
    int queue[MAX_VERTICES];
    int front=0, rear=0;
    visited[s] = true;
    queue[rear++] = s;
    while(front!=rear)
    {
        s = queue[front++];
        printf("%d ", s);
        for(int adj=0; adj<g->V; adj++)
        {
            if(g->adj[s][adj] && !visited[adj])
            {
                visited[adj] = true;
                queue[rear++] = adj;
            }
        }
    }
}
```

```
int main()
{
    int numVertices;
    printf("Enter the number of vertices in the
            graph : ");
    scanf("%d", &numVertices);

    Graph *g = Graph_create(numVertices);

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

    printf("Enter the edges(vertex1 vertex2):\n");
    for(int i=0; i<numEdges; i++){
        int v, w;
        scanf("%d %d", &v, &w);
        Graph_addEdge(g, v, w);
    }
    int startVertex;
    printf("Enter the starting vertex for BFS:");
    scanf("%d", &startVertex);

    printf("Following is Breadth First Traversal
            (starting from vertex %d)\n", startVertex);
    Graph_BFS(g, startVertex);

    Graph_destroy(g);
    return 0;
}
```

Output:

Enter the number of vertices in the graph: 4
Enter the number of edges in the graph: 6
Enter the edges
0 1
0 2
1 2
2 0
2 3
3 3
Enter the starting vertex for BFS: 2
Following is Breadth First Traversal
2 0 3 1

Date / /
Page

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

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

#define MAX_NODES 100
#define MAX_EDGES 100

int graph[MAX_NODES][MAX_NODES];
int visited[MAX_NODES];

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

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

```
    return 1;
}
int main()
{
    int n, m;
    printf("Enter the number of nodes and edges:");
    scanf("%d %d", &n, &m);

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

Output:

```
Enter number of nodes and edges: 4 6
Enter edges:
0 1
0 2
2 3
2 4
2 5
```

The graph is connected.

1. Hacker rank

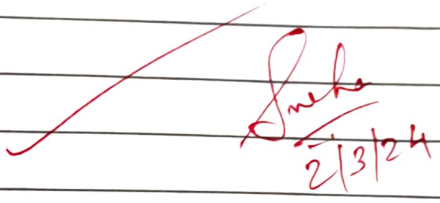
```
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Node{
  int data;
  struct Node *left;
  struct Node *right;
} Node;

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

void inOrder (struct Node *root, int *result, int *index)
{
        if (root == NULL) return;
        if (level % k == 0)
        {
         Node *temp = root -> left;
         root -> left = root -> right;
         root -> right = temp;
        }
       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);
}

int** swapNodes(int indexes_rows, int colums, int queries)
{
Node** nodes = (Node**) malloc(indexes_rows+1) * sizeof(node*)
for(int i=1; i<= indexes_rows; i++) {
        nodes[i] = createNode(i);
}
for(int i=0; i<indexes_rows; i++)
{
    int left = indexes[i][0];
    int right = indexes[i][1];
    if(leftIndex != -1) nodes[i+1]->left = nodes[leftIndex];
    if(rightIndex != -1) nodes[i+1]->right = nodes[rightIndex];
}

int **result = (int**) malloc(queries * sizeof(int*));
*result_rows = queries;
*result_colums = indexes_rows;
for(int i=0; i<queries; i++)
{
    swapAtLevel(nodes[1], queries[i], 1);
    int *traversal = (int*) malloc(indexes_rows * sizeof(int
    int index=0;
    inOrder(nodes[1], traversalResult, &index);
    result[i] = traversalResult;
}
```

```
    free(nodes);
    return result;
}
int main(){
  int n;
scanf("%d", &n);

int ** indexes = malloc(n * sizeof(int*));
for(int i=0; i<n; i++)
{
    indexes[i] = malloc (2 * sizeof(int));
    scanf("%d %d", & indexes[i][0], &indexes[i][1]);
}

int queries;
scanf("%d", &queries)
 for(int i=0;i< queries, i++)
 {
    scanf("%d", &queries[i]);
}
  int result_rows;
  int result_columns;
 int ** result = swapNodes(n, 2, indexes, queries, &result_rows
& result_columns);

for(int i=0; i< result_rows; i++){
   for(int j=0; j<result_columns; j++)
   {
       printf("%d", result[i][j]);
   }
    printf("\n");
    free(result[i]);
  }
```

```
        free (result[i]);
    }
    free (result[i]);
}
free (result);

for (int i=0; i<n; i++)
{
    free (indexes[i]);
}
    free (indexes);
    free (queries);
    return 0;
}
```

Output.



29-2-24

Date __/__/__
Page _____

LAB - 10

1. Hashing with Linear Probing

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define Table_size 100
#define key_length 5
#define MAX_NAME_LENGTH 50
#define MAX_DESIGNATION 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 hash_function(const char * key, int m)
{
    int sum=0;
    for(int i=0; key[i] != '\0'; i++)
    {
        sum += key[i];
    }
    return sum % m;
}
```

```
void insert(struct HashTable *ht, struct Employee *
  emp)
{
   int index = hash_function(emp->key, TABLE_SIZE);
   while(ht->table[index] != NULL)
   {
      index = (index + 1) % TABLE_SIZE;
   }
      ht->table[index] = emp;
   }
   struct Employee* search(struct HashTable *ht, const
                                        char *key)
   {
    int index = hash_function(key, TABLE_SIZE);

    while(ht->table[index] ~~key~~ != NULL)
    {
      if(strcmp(ht->table[index]->key, key)==0)
      {
            return ht->table[index];
      }
      index = (index + 1) % TABLE_SIZE;
    }
    return NULL;
}

int main(){
   struct HashTable ht;
   struct Employee *emp;
   char key[KEY_LENGTH];
   FILE *file
```

```
char filename[100];
char line[100];
for (int i=0; i<TABLE_SIZE; i++)
{
    ht.table[i]=NULL;
}
printf("Enter the filename containing employee records");
scanf("%s",filename);
file = fopen(filename, "r");
if (file==NULL)
{
  printf("Error opening file \n");
  return 1;
}
while (fgets(line, sizeof(line), file))
{
  emp=(struct Employee *) malloc(sizeof(struct
                                  Employee));
  sscanf(line, "%s %s %s %f", emp->key, emp->name,
                  emp->designation, &emp->salary);
  insert(&ht, emp);
}
fclose(file);
printf("Enter key to search:");
scanf("%s", key);
emp= search(&ht, key);
if (emp != NULL)
{
  printf("Employee record found with key %s:\n",
                                  emp->key);
printf("Name: %s\n", emp->name);
printf("Designation: %s", emp->designation);
```

```
        printf("Salary : %.2f \n", emp->salary);
    }
    else
    {
        printf("Employee record not found for key %d", key);
    }
    for (int i=0; i< TABLE_SIZE ; i++)
    {
        if (ht.table[i] != NULL)
        {
            free(ht.table[i]);
        }
    }
    return 0;
}
```

Output:

Enter the number of memory locations (m) : 2
Enter the number of employee records (n) : 2

Enter details for Employee 1:
Enter the key : 2034

Enter details for Employee 2:
Enter the key : 1024

Hash Table

Index	key
1	2034
2	1024

2/3/24