INDEX

NAME: PRADEEP PT STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: DS LAB

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	7/12/2023	Practice Programs.	1 (10)	SK 7/12
2.	21/12/2023	Stack Implementation	3	SK 21/12
3.	28/12/2023	Infix to Postfix, Postfix Evaluation, linear queue.	10 (10)	SK 28/12
4.	11/01/2024	Circular queue, singly linked list	19 (10)	SK 11/1/24
5.	18/01/2024	singly linked list (Deletion) Leetcode - (1) Leetcode - (2)	27 (10)	SK 18/1/24
6.	25/01/2024	Linked list operations, stack and queue implementation using ll	37 (10)	SK 25/1/24
7.	01/02/2024	Doubly Linked List Leetcode - (3)	50 (10)	SK 1/2/24
8.	15/02/2024	Binary Search Tree Leetcode - (4)	58 (10)	SK 15/2/24
9.	22/02/2024	BFS and DFS HackerRank Problem	64 (10)	SK 22/2/24
10	29/02/2024	Hashing	68 (10)	SK 29/2/24

10

SK 29/2/24

7/12/23

LAB 1 : Practice Programs (output)

i)

1. Create Account
2. Deposit
3. Withdraw
4. Balance Inquiry
5. Exit

Enter your choice : 3
Enter the amount to withdraw : 5000
Insufficient funds!

ii)

Enter the number of strings (up to 10) : 6
Enter 6 strings :
Geography
Algorithm
Detection
Support
Payment
Journey

Sorted strings lexicographically :
Algorithm
Detection
Geography
Journey
Payment
Support

iii)

Enter the elements to check if it is present in the array :
1 2 3
4 5 6
7 8 9
6

Element 6 is present in the array

iv) Enter the larger string : environment
Enter the substring to search for : men
substring found at index 7 in the larger string

v) Enter the elements
1 2 3 4 2 6 11 8
Enter the number to find the last occurrence : 2
The last occurence of 2 is at index 4

vi) Enter the array elements
2 5 7 11 13 19 29
Enter the element to search : 13
Element 13 is at 5 position

vii) Enter the array elements.
6 8 12 22 36 48
Enter the element to search 12
Element 12 is at 3 position

viii) Enter the elements
4 7 2 9 1 8 3 6
Maximum Element : 9
Minimum Element : 1

21/12/23

LAB-②:

Program-1: Swapping of two numbers using pointers

```
#include <stdio.h>
void swap (int *ptr1, int *ptr2);
void main()
{
int x, y;
printf("Enter the 1st number\n");
scanf("%d", &x);
printf("Enter the 2nd number\n");
scanf("%d", &y);
printf("Numbers before swapping: x=%d
y=%d\n", x, y);
swap(&x, &y);
printf("Numbers after swapping: x=%d
y=%d\n", x, y);
}
void swap(int *ptr1, int *ptr2)
{
int temp = *ptr1;
*ptr1 = *ptr2;
*ptr2 = temp;
}
```

Output:

```
Enter the 1st number
20
Enter the 2nd number
40
Numbers before swapping: x=20 y=40
Numbers after swapping: x=40 y=20
```

Program ②: Stack implementation

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int top = -1;
int a[SIZE];
void push();
void pop();
void display();
void main()
{
 int option;
 while(1){
 printf("Enter the option for the following operations:\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
 scanf("%d", &option);
 switch(option)
 {
  case 1: push();
  break;
  case 2: pop();
  break;
  case 3: display();
  break;
  case 4: exit(0);
  default: printf("Invalid input\n");
  break;
 }
 }
}
void push()
{
 int x;
 if(top == SIZE-1)
```

```
{
printf ("Overflow\n");
}
else
{
printf ("Enter the element do be added\n");
scanf ("%d", &x);
top = top + 1;
a[top] = x;
}
}
void pop()
{
if (top == -1)
{
printf ("Underflow\n");
}
else
{
printf ("Popped element is %d\n", a[top]);
top = top - 1;
}
}
void display()
{
int i;
if (top == -1)
{
printf ("stack is empty\n");
}
else {
for (i = top; i >= 0; --i)
printf ("%d \n", a[i]);
}
}
```

Output:

```
Enter the option for the following operations:
1. Push
2. Pop
3. Display
4. Exit
1
Enter the element to be added
25
Enter the option for the following operations:
1. Push
2. Pop
3. Display
4. Exit
2
Popped element is 25
Enter the option for the following operations
1. Push
2. Pop
3. Display
4. Exit
3
Stack is empty.
Enter the option for the following operations:
1. Push
2. Pop
3. Display
4. Exit
4
```

Sathya
21/12

Program ③ : Dynamic memory allocation

```
#include <stdio.h>
int main()
{
int *ptr;
int n;
printf("Enter number of elements : \n");
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]);
}
}
free(ptr); 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
using realloc\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<n1; ++i){
printf("%d", ptr[i]);
}
free(ptr);
printf("Memory Freed\n");
}
}
```

Output:

Enter number of elements : 5
Memory successfully allocated using malloc.
Enter elements of array
2
4
6
8
10
The elements of the array are : 2, 4, 6, 8, 10, Memory Freed
Memory successfully allocated using calloc
Enter elements of array
2
4
6
8
10
The elements of the array are : 2, 4, 6, 8, 10
Enter new size for realloc.
6
Memory successfully re-allocated using realloc.
Enter more variables:
12
The elements of the array are : 2, 4, 6, 8, 10, 12 Memory Freed.

28/12/23

Lab ③:

Program 1: Conversion of infix expression to postfix expression

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAX 100
char st[MAX];
int top = -1;
void push (char st[], char);
char pop (char st[]);
void InfixtoPostfix (char a[], char b[]);
int Priority (char);
int main()
{
char infix[100], postfix[100];
printf ("Enter any expression\n");
gets (infix);
InfixtoPostfix (infix, postfix);
printf ("The correrponding postfix expression
is: ");
puts (postfix);
return 0;
}
void InfixtoPostfix (char a[], char b[])
{
int i=0, j=0;
char temp;
while (a[i] != '\0')
{
if (a[i] == '(')
{
push (st, a[i]);
```

```
i++;
}
else if (a[i] == ')')
{
while ((top != -1) && (st[top] != '('))
{
b[j] = pop(st);
j++;
}
if (top == -1)
{
printf("Incorrect expression");
exit(1);
}
temp = pop(st);
i++;
}
else if (isdigit(a[i]) || isalpha(a[i]))
{
b[j] = a[i];
i++;
j++;
}
else if (a[i] == '+' || a[i] == '-' || a[i] == '*'
|| a[i] == '/' || a[i] == '%')
{
while ((top != -1) && (st[top] != '(') && (
Priority(st[top]) > Priority(a[i])))
{
  b[j] = pop(st);
j++;
}
push(st, a[i]);
i++;
}
```

```
else
{
printf("Incorrect element in Expression");
exit(1);
}
}
while((top!=-1)&&(st[top]!='('))
{
target b[j]=pop(st);
j++;
}
b[j]='\0';
}
int Priority(char op)
{
if(op=='/'||op=='*'||op=='%')
return 1;
else if(op=='+'||op=='-')
return 0;
}
void push(char st[], char val)
{
if(top==MAX-1)
printf("Stack Overflow");
else
top++;
st[top]=val;
}
}
char pop(char st[])
{ if(top==-1)
printf("Stack Underflow");
else
{
```

```
val = st[top];
top--;
}
return val;
}
```

output: Enter any expression:
A*B+C*D-E
The corresponding postfix expression is:
AB*CD*+E-

Program 2: Postfix Evaluation

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int a[MAX];
int top = -1;
int main()
{
char expression[50];
printf("Enter any expression\n");
gets(expression);
int result = evaluate(expression);
printf("Result = %d\n", result);
return 0;
}
void push(int x)
{
if (top >= MAX-1)
printf("Stack Overflow\n");
else
{
top++;
a[top] = x;
}
```

```
int pop()
{
if (top < 0)
{
printf("Stack Underflow\n");
}
else
{
int x = a[top];
top--;
return x;
}
int operator (char symbol)
{
if (symbol == '+' || symbol == '-' || symbol
== '*' || symbol == '/')
{
return 1;
}
return 0;
}
int evaluate (char expression)
{
int i = 0;
char symbol = expression[i];
int op1, op2, result;
while (symbol != '\0')
{
if (symbol >= '0' && symbol <= '9') {
int num = symbol - '0';
push(num);
}
else if (operator(symbol))
{
```

```
op2 = pop();
op1 = pop();
switch (symbol) {
case '+': result = op1 + op2;
break;
case '-': result = op1 - op2;
break;
case '*': result = op1 * op2;
break;
case '/': result = op1 / op2;
break;
}
push(result);
}
i++;
symbol = expression[i];
}
result = pop();
return result;
}
```

Output:

Enter the expression

5 6 7 + * 8 -

Result = 57.

8/12

8/12

Lab (4)

11/01/2024

Program 3a. Linear Queue Implementation

```
#include <stdio.h>
#include <string.h>
#define MAX 100
int rear = -1, front = -1;
int queue[MAX];
void enqueue();
void display();
int dequeue();
int main()
{
 int option, val;
do
{
printf("\n Enter an option to perform following operations \n 1. Insert \n 2. Delete \n 3. Display \n 4. Exit \n");
scanf("%d", &option);
switch (option)
{
case 1: enqueue();
break;
case 2: val = dequeue();
printf("Element deleted from queue is %d", val);
break;
case 3: display();
break;
}
}
while (option != 4);
return 0;
}
```

```
void enqueue()
{
int x;
printf("Enter the number to be inserted
in the queue\n");
scanf("%d", &x);
if (rear == MAX-1)
printf("Overflow");
else if (front == -1 && rear == -1)
front = rear = 0;
else
rear = rear+1;
queue[rear] = x;
}
int dequeue()
{
int y;
if (front == -1 || front > rear)
printf("Underflow");
else
{
y = queue[front];
front = front+1;
}
return y;
}
void display() {
int i;
printf("Elements in the queue:\n");
if (front == -1 || front > rear)
printf("Underflow");
for (i = front; i <= rear; i++)
printf("\t%d", queue[i]);
}
```

Output:

```
Enter an option to perform following
operations
1. Insert
2. Delete
3. Display
4. Exit
1
Enter the number to be inserted in the queue
18

Enter an option to perform following operations
1. Insert
2. Delete
3. Display
4. Exit
2
Element deleted from queue is : 18

Enter an option to perform following operations
1. Insert
2. Delete
3. Display
4. Exit
3

Elements in the queue :
Underflow
```

SK 11/1/24

Program 3b. Circular Queue Implementation

```
#include <stdio.h>
#define MAX 5
int rear = -1, front = -1;
int queue[MAX];

void enqueue();
int dequeue();
void display();
int main()
{
int option, value;
do {
printf("\n Enter an option to perform the following operations: \n 1. Insert \n 2. Delete \n 3. Display \n 4. Exit \n");
scanf("%d", &option);

switch(option){
case 1: enqueue();
break;
case 2: value = dequeue();
printf("Element deleted from queue: %d \n", value);
break;
case 3: display();
break;
}
} while(option != 4);
return 0;
}
void enqueue()
{
```

```
int x;
printf ("Enter the element to be inserted:");
scanf ("%d", &x);
if ((rear+1)%MAX == front)
{
printf ("Overflow. Queue is full \n");
}
else if (front == -1 && rear == -1) {
front = rear = 0;
queue [rear] = x;
} else {
rear = (rear+1) % MAX;
queue [rear] = x;
}
}
int dequeue () {
int y;
if (front == -1 && rear == -1) {
printf ("Underflow. Queue is empty \n");
return -1;
}
y = queue [front];
if (front == rear) {
front = rear = -1;
} else {
if (front == MAX-1) {
front = 0;
}
else {
front = (front+1) % MAX;
}
}
return y;
}
```

```
void display () {
int i;
printf ("Elements in the Queue : \n");
if (front == -1 && rear == -1) {
printf ("Underflow. Queue is empty \n");
}
else {
if (front <= rear) {
for (i = front ; i <= rear ; i++) {
printf ("\t %d", queue[i]);
}
else {
for (i = front ; i < MAX ; i++) {
printf ("\t %d", queue[i]);
}
for (i = 0 ; i <= rear ; i++) {
printf ("\t %d", queue[i]);
}
}
printf ("\n");
}
}
```

Output:

```
Enter an option to perform the following operation
1. Insert
2. Delete
3. Display
4. Exit
1
Enter the element to be inserted : 78

Enter an option to perform the following operation
```

1. Insert
2. Delete
3. Display
4. Exit

1

Enter the element to be inserted : 100

Enter an option to perform the following operations:

1. Insert
2. Delete
3. Display
4. Exit

3

Elements in the queue:

78 100

Enter an option to perform the following operations:

1. Insert
2. Delete
3. Display
4. Exit

2.

element deleted from queue : 78

Enter an option to perform the following operations

1. Insert
2. Delete
3. Display
4. Exit

4

Program 4: Singly Linked List Implementation

```
#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();

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

```
default: printf("Exiting the program");
return 0;
}
}
}
void push() {
Node * temp = (Node*) malloc (sizeof(Node));
int new_data;
printf("Enter data in the new node: ");
scanf("%d", &new_data);
temp -> data = new_data;
temp -> next = head;
head = temp;
}
void append() {
Node * temp = (Node*) malloc (sizeof(Node));
int new_data, pos;
printf("Enter data in the new node: ");
scanf("%d", &new_data);
printf("Enter position of the new node: ");
scanf("%d", &pos);
temp -> data = new_data;
temp -> next = NULL;
if (pos == 0) {
temp -> next = head;
head = temp;
return 0;
}
Node * temp1 = head;
while (pos--) {
temp1 = temp1 -> next;
}
Node * temp2 = temp1 -> next;
temp -> next = temp2;
temp1 -> next = temp;
}
```

insert

```
void append () {
Node * temp = (Node *) malloc (sizeof (Node));
int new_data;
printf ("Enter data in the new node :");
scanf ("%d", &new_data);
temp -> data = new_data;
temp -> next = NULL;
if (head == NULL) {
 head = temp;
 return;
}
Node * temp1 = head;
while (temp1 -> next != NULL) {
 temp1 = temp1 -> next;
}
temp1 -> next = temp;
}

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

Output:

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

Enter choice : 1
Enter data in the new node : 18

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

Enter choice : 2.
Enter data in the new node : 78

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

Enter choice : 4

18 -> 78 -> NULL

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

Enter choice : 3
Enter data in the new node : 64
Enter position of the new node : 2

Week 5:

Program 5: Singly linked list (Deletion)

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

struct node
{
int data;
struct node *next;
};
struct node *start = NULL;
void create();
void delete_begin();
void delete_end();
void delete_pos();

void main()
{
int option;
do
{
printf("Enter an option to perform following operations: \n 1. Create a linked list \n 2. Delete from beginning \n 3. Delete at end \n 4. Delete at any position \n 5. Display \n 6. Exit \n");
scanf("%d", &option);
switch(option)
{
case 1: create();
        printf("Linked list created successfully \n");
        break;
```

```
case 2: delete_begin();
        printf("Element deleted\n");
        break;
case 3: delete_end();
        printf("Element deleted\n");
        break;
case 4: delete_pos();
        printf("Elements deleted\n");
        break;
case 5: printf("Elements in linked list\n");
        display();
        break;
}
} while(option!=6);
}

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

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

void delete_end()
{
struct node *ptr, *preptr;
if (start == NULL)
printf ("Linked list is empty");
ptr = start;
while (ptr -> next != NULL)
{
preptr = ptr;
ptr = ptr -> next;
}
preptr -> next = NULL;
free (ptr);
}
```

```
void delete_pos()
{
struct node *ptr, *preptr, *postptr;
int pos, count = 1;
printf("Enter the position\n");
scanf("%d", &pos);
if (start == NULL)
printf("Linked list is empty");
ptr = start;
if (pos == 1)
{
start = start -> next;
free(ptr);
}
else
{
while (count < pos && ptr != NULL)
{
preptr = ptr;
ptr = ptr -> next;
postptr = ptr -> next;
count++;
}
if (pos == count)
{
preptr -> next = postptr;
free(ptr);
}
}
}

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

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

Output:

```
Enter an option:
1. Create a linked list
2. Delete from beginning
3. Delete at end
4. Delete at any position
5. Display
6. Exit
1
Enter -1 to exit
Enter the data: 2
Enter the data: 4
Enter the data: 6
Enter the data: 8
Enter the data: -1
Linked list created successfully.

Enter an option
1. Create a linked list
2. Delete from beginning
3. Delete at end
4. Delete at any position
5. Display
6. Exit
2
```

Element deleted:
5
Elements in the list:
4 6 8

Enter an option
1. Create
2. Delete from beginning
3. Delete at end
4. Delete at any position
5. Display
6. End.

4
Enter the position : 2
Element deleted.

5.
Elements in the list
4 8

Enter an option
1. Create
2. Delete from beginning
3. Delete at end
4. Delete at any position
5. Display
6. End

3.
Element deleted
5
Elements in the list
4

Leetcode 1: min-stack:

```
#include <stdio.h>
#include <stdlib.h>
#define max 4

typedef struct {
  int top;
  int st[max];
  int min[max];
} Minstack;

Minstack * minstackCreate() {
Minstack * stack = (Minstack *) malloc
(sizeof(Minstack));
stack -> top = -1;
return stack;
}
void minstackPush (Minstack * obj, int val)
{
if (obj -> top == max-1)
{
printf ("Stack Full \n");
return;
}
obj -> st [++ obj -> top] = val;
if (obj -> top > 0)
{
if (obj -> min [obj -> top-1] < val)
obj -> min [obj -> top] = obj -> min [obj -> top-1];
else
  obj -> min [obj -> top] = val;
}
```

```
void minStackPop (MinStack * obj)
{
if (obj->top == -1)
 {
  printf ("Stack empty \n");
  return;
}
else
{ obj->top--;
}
}
int minStackTop (MinStack * obj) {
 if (obj->top == -1)
{
printf("Stack empty \n");
return -1;
}
return obj->st[obj->top];
}
int minStackGetMin (MinStack * obj) {
if (obj->top == -1)
{
printf ("min stack empty \n");
return -1;
}
return obj->min[obj->top];
}
void minStackFree (MinStack * obj)
free(obj);
int main()
{
MinStack * obj = minStackCreate();
```

```
minStackPush(obj, 3);
minStackPush(obj, 1);
minStackPush(obj, 2);
minStackPush(obj, 1);
printf("Min: %d\n", minStackGetMin(obj));
printf("Top: %d\n", minStackTop(obj));
printf("Min: %d\n", minStackGetMin(obj));
minStackFree(obj);
}
return 0;
```

Output:

```
Min: 1
Top: 1
Min: 2
```

18/1/24

LeetCode 2: Reverse Linked List II

```
struct ListNode * reverseBetween(struct
ListNode * head, int left, int right)
{
struct listnode * ptr1 = head;
int temp = left - 1;
while (temp--)
{
ptr1 = ptr1 -> next;
}
int count = right - left + 1;
int *a = (int *) malloc(count * sizeof(int));
for (i = 0; i < count; i++)
{
a[i] = ptr1 -> val;
```

```
ptr1 = ptr1->next;
}
struct ListNode * ptr = head;
left--;
while (left--) {
printf ("%d", ptr->val);
ptr = ptr->next;
}
for (int i = count-1; i > -1; i--) {
ptr->val = a[i];
ptr = ptr->next;
}
return head;
}
```

Output:

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

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

25/01/24

Week 6:

Lab Program 1: Linked list operations (sort, Reverse and Concatenate)

```
#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;
void create();
struct node *create2(struct node *);
void sort();
struct node *concatenate(struct node *,
struct node *);
void reverse();
void display(struct node *);
void display2(struct node *);

int main()
{
 int option;
 struct node *a = NULL;
 do {
 printf("\n ***** MAIN MENU ***** \n\n
1. create a linked list \n 2. Create two
linked lists for concatenation \n 3. sort
\n 4. Concatenate \n 5. Reverse \n 6. Display
linked list \n 7. Display Concatenated
linked list \n 8. Exit \n");
```

```
printf("\n Enter an option to perform the
above operations : ");
scanf("%d", &option);
switch (option);
{
case 1: start = create(start);
        printf("\n Linked list created
                successfully \n");
        break;
case 2: printf("\n Linked list 1: \n");
        s1 = create2(s1);
        printf("\n Linked list 2: \n");
        s2 = create2(s2);
        printf("\n Linked lists created
                successfully \n");
        break;
case 3: sort();
        printf("\n Linked list sorted \n");
        break;
case 4: a = concatenate(s1, s2);
        printf("\n Linked lists concatenated
                successfully \n");
        break;
case 5: reverse();
        printf("\n Linked list reversed \n");
        break;
case 6: printf("\n Elements in the linked
        list \n");
        display(start);
        break;
case 7: printf("\n Elements in the linked
        list after concatenation \n");
        display2(a);
        break;
```

```
}
} while (option != 8);
return 0;
}
struct node *
void 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;
}
start = prev;
}
void display ( struct node * p )
{
 void . struct node * ptr;
  ptr = p;
  while ( ptr != NULL )
  {
  printf ( "\t %d ", ptr -> data );
  ptr = ptr -> next;
}
print f ( "\n" );
}
```

Output:

++++ MAIN MENU ++++

1. Create a linked list
2. Create two linked lists for concatination
3. Sort
4. Concatenate
5. Reverse
6. Display linked list
7. Display concatenated linked list
8. Exit

Enter an option to perform the above operations : 1

Enter -1 to exit

Enter the data: 2
Enter the data: 4
Enter the data: 6
Enter the data: 8
Enter the data: 10
Enter the data: -1

Linked list created successfully
Enter an option: 5

Linked list reversed.
Enter an option: 6

Elements in the linked list
10 8 6 4 2

Enter an option: 3
Linked list sorted

Elements in the linked list
2 4 6 8 10

Enter an option: 2
Linked list 1:
Enter -1 to exit

Enter the data: 18
Enter the data: 78
Enter the data: 64
Enter the data: -1

Linked list 2:
Enter -1 to exit

Enter the data: 24
Enter the data: 84
Enter the data: -1

Linked list created successfully
Enter an option: 4

Linked list concatenated successfully
Enter an option: 7

Elements in the linked list after concatenation
18 78 64 24 84
Enter an option: 8

Lab Program 2:
1) Stack Implement using linked list

```
#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 following operations\n 1. Push \n 2. Pop \n 3. Display \n 4. Exit \n");
 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_node = (struct node*) malloc(sizeof(struct node));
 new_node -> data = num;
 new_node -> next = start;
 start = new_node;
}
void pop()
{
 struct node *ptr;
```

```
ptr = start;
if (start == NULL)
{
printf ("Stack is empty\n");
exit(0);
}
else
{
 ptr = start;
 start = ptr -> next;
 printf ("\n Element popped from the stack
  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 : 7

Enter the number to perform following operation
1

Enter the data : 4

Enter the number.
3

4 2

Enter the number :
2

Element popped from the stack : 4

Enter the number :
3

Elements in the stack
2

Enter the number to perform following operations
1. Push
2. Pop
3. Display
4. Exit

2

Element popped from the stack : 2

Enter the number to perform following operations.
4

ii) Stack . Queue . Implementation using linked list.

```
#include <stdio.h>
#include <stdio.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 the number to perform following operations \n 1. Enqueue \n 2. Dequeue \n 3. Display \n 4. Exit \n");
 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 the 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
{
while (ptr -> next != NULL)
{
preptr = ptr;
ptr = ptr -> next;
}
preptr -> next = NULL;
printf ("\nElement popped from the
stack 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 operation
1. Enqueue
2. Dequeue
3. Display
4. Exit
1

Enter the data
18
Enter the number
1.
Enter the data
78

Enter an option
1. Enqueue
2. Dequeue
3. Display
4. Exit
3
```

78 18

Enter an option
2
Element popped from the stack is : 18

Enter an option
3

78

Enter an option
1. Enqueue
2. Dequeue
3. Display
4. Exit
4

Sakthi 25th/24

01/02/24 Week 7

Lab Program 1: Doubly linked list
a) Create a doubly linked list
b) Insert a new node to the left of the node
c) Delete the node based on a specific value
d) Display the contents of the list

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

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

```
struct node *start = NULL;
void create();
void insert();
void delete();
void display();

void main()
{
int option;
do
{
printf("\n** MAIN MENU **\n\n1. Create a doubly linked list \n2. Insert at left
\n3. Delete (Specific value) \n4. Display
\n5. Exit\n\nEnter an option: ");
scanf("%d", &option);
switch(option)
{
case 1: create();
        printf("\nDoubly linked list created\n");
        break;
case 2: insert();
        printf("\nNode inserted\n");
        break;
case 3: delete();
        printf("\nNode deleted\n");
        break;
case 4: printf("\nElements in the
        doubly linked list\n");
        display();
        break;
}
} while(option != 5);
}
```

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

```
void insert()
{
struct node *new_node, *ptr;
int pos, val, count = 0;
printf("\n Enter the data: ");
printf(" scanf("%d", &val);
printf("\n Enter the position before which
the data has to be inserted: ");
scanf("%d", &pos);
new_node = (struct node *) malloc (sizeof (struct
node));
new_node -> data = val;
ptr = start;
while (count < pos - 1 && ptr != NULL)
{
 ptr = ptr -> next;
 count++;
}
if (count == pos - 1 && ptr != NULL) {
 new_node -> next = ptr;
 if (ptr -> prev != NULL)
 {
  new_node -> prev = ptr -> prev;
  ptr -> prev -> next = new_node;
 }
 else
 {
  start = new_node;
  new_node -> prev = NULL;
 }
 ptr -> prev = new_node;
}
else
{
```

```
printf("Invalid position\n");
free(new_node);
}
}

void delete()
{
struct node *ptr;
int num;
printf("Enter the data to be deleted\n");
scanf("%d", &num);
ptr = start;
while(ptr != NULL && ptr->data == num)
{
start = ptr->next;
if(start != NULL)
start->prev = NULL;
free(ptr);
ptr = start;
}
while(ptr != NULL && ptr->data != num)
ptr = ptr->next;
if(ptr == NULL)
{
printf("Data not found");
}
else
{
if(ptr->prev != NULL)
ptr->prev->next = ptr->next;
if(ptr->next != NULL)
ptr->next->prev = ptr->prev;
free(ptr);
}
}
```

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

Output:

```
***MAIN MENU***

1. Create a doubly linked list
2. Insert at left
3. Delete (specific value)
4. Display
5. Exit

Enter an option : 1

Enter -1 to end
Enter the data : 2
Enter the data : 4
Enter the data : 6
Enter the data : -1

Doubly linked list created

Enter an option : 2
Enter the data : 8
Enter the position before which the
data has to be inserted : 2
```

Node inserted

Enter an option : 4

Elements in the doubly linked list
2 8 4 6

Enter an option : 3
Enter the data to be deleted : 4

Node deleted.
Enter an option : 4

Elements in the doubly linked list
2 8 6

* Leetcode 3 : Split Linked List in Parts.

```
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
struct Listnode* ptr = head;
*returnSize = k;
int count = 0;

while(ptr != NULL) {
count++;
ptr = ptr -> next;
}
int num = count / k, a = count % k;

struct ListNode **L = (struct ListNode**)
calloc(k, sizeof(struct ListNode*));

ptr = head;
```

```
for (int i=0; i<k; i++) {
  L[i]= ptr;
int segmentSize = nums + (a-->0?1:0);
for (int j=1; j< segmentSize; j++) {
  ptr = ptr -> next;
}
if (ptr!= NULL) {
struct ListNode * next = ptr -> next;
ptr -> next = NULL;
ptr = next;
}
}
return L;
}
```

Output:

i) Input : Lead = [1, 2, 3], k=5
Output : [[1], [2], [3], [], []]

ii) Input : Lead = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], k=3
Output : [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]]

15/02/24 Week (8):

Lab Program: Write a program

i) To construct a binary search tree

ii) To traverse the tree using all the methods i.e., in-order, preorder, and postorder.

iii) To display the elements in the tree.

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

struct node
{
 struct node *left;
 struct node *right;
 int data;
};
 struct node *tree = NULL;

void create();
void pre (struct node *);
void post (struct node *);
void in (struct node *);

void main()
{
 int option;
 do
 {
 printf ("\n\n***MAIN MENU***\n\n1. Create a binary search tree \n2. Preorder traversal \n3. Postorder traversal \n4. Inorder traversal \n5. Exit \n\n Enter an option: ");
 scanf ("%d", &option);
 switch (option)
```

```
{
case 1 : create();
         printf("Binary search tree created\n");
         break;
case 2 : printf("\n The elements in the
         tree are \n");
         pre(tree);
         break;
case 3 : printf("\nThe elements in the
         tree are \n");
         post(tree);
         break;
case 4 : printf("\n The elements in the
         tree are \n");
         break;
}
} while (option != 5);

void create()
{
 int val;
 printf("\n Enter -1 to end");
 printf("\n Enter the element:");
 scanf("%d", &val);
 while (val != -1)
 {
  struct node *ptr, *nodeptr, *parentptr;
  ptr = (struct node *) malloc (sizeof(struct
  node));
  ptr -> data = val;
  ptr -> left = NULL;
  ptr -> right = NULL;
  if (tree = NULL)
  {
```

```
        tree = ptr;
        tree -> left = NULL;
        tree -> right = NULL;
    }
    else
    {
        parent ptr = NULL;
        nodeptr = tree;
        while (nodeptr != NULL)
        {
            parent ptr = node ptr;
            if (val < nodeptr -> data)
                node ptr = nodeptr -> left;
            else
                nodeptr = nodeptr -> right;
        }
        if (val < parentptr -> data)
            parentptr -> left = ptr;
        else
            parentptr -> right = ptr;
        printf ("\n Element : Enter the element : ");
        scanf ("%d", &val);
    }
}

void pre (struct node * tree)
{
    if (tree != NULL)
    {
        printf ("%d \t", tree -> data);
        pre (tree -> left);
        pre (tree -> right);
    }
}
```

```
void in(struct node *tree)
{
if(tree!=NULL)
{
printf("%c",in(tree->left);
        printf("%d",tree->data);
        in(tree->right);
}
}
void post(struct node *tree)
{
 if(tree!=NULL)
{
   post(tree->left);
   post(tree->right);
   printf("%d",tree->data);
}
}
```

Output:

```
***Main Menu***

1. Create a binary search tree
2. Preorder traversal
3. Postorder traversal
4. Inorder traversal
5. Exit

Enter an option : 1

Enter -1 to end

Enter the element : 8 1 5 3 9 4 6 7
-1

Binary search tree created.
```

Enter an option : 2

The elements in the tree are
8 1 5 3 4 6 7 8

Enter an option : 3

The elements in the tree are
4 3 7 6 5 1 9 8

Enter an option : 4

The elements in the tree are
1 3 4 5 6 7 8 9
Enter an option : 5

* LeetCode 4 : Rotate list

```
struct listnode * rotateRight (struct ListNod
* head, int k) {
  struct ListNode *ptr, *ptr1;
  int count = 0, num;
  if (head == NULL || head -> next == NULL) {
    return head;
  }
  ptr = head;
  while (ptr -> next != NULL)
  {
    count++;
    ptr = ptr -> next;
  }
  num = k % (count + 1);
  while (num--)
  {
```

```
ptr = head;
while (ptr -> next != NULL)
{
ptr1 = ptr;
ptr = ptr -> next;
}
ptr -> next = head;
ptr1 -> next = NULL;
head = ptr;
}
return head;
}
```

Output:

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

k = 2

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

head = [0, 1, 2], k = 4

output 2 = [2, 0, 1]

8/2

22/2

22/02/24 Week (9)

Lab Program: 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 <stdio.h>
#define MAX_VERTICS 50
typedef struct Graph_t {
    int V;
    int adj[MAX_VERTICES][MAX_VERTICS];
}
Graph;
int DFS_V[50];
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]=0;
   }
}

 return g;
}
void Graph_add Edge (Graph *g, int v, int w)
{
 g-> adj[v][w] = 1;
 g-> adj[w][v] = 1;
}
void BFS (Graph * g, int root) {
 int visited [g->V+1];
```

```
for (int i=0; i<=g->V; i++)
visited [i]=0;
int queue [g->V+1];
int front = 0, rear = 0;
visited [root]=1;
queue [root]=1;
while (front != rear)
{
  root = queue [front++];
printf ("%d", root);
for (int i=0; i<g->V; i++)
{
  if (g->adj [root][i] == 1 && visited [i] != 1)
{
  visited [i] = 1;
  queue [rear++] = i;
}
}
}
}

int DFS (Graph *g, int root) {
  for (int i=0; i<=g->V; i++)
{
  if (g->adj [root][i] == 1 && DFS_V [i] != 1)
{
  DFS_V [i] = 1;
  DFS (g, i);
}
}

  int count = 0;
  for (int i=0; i<g->V; i++) {
  if (DFS_V [i] == 1) {
    count++;
  }
} return count;
}
```

```
int main()
{
  Graph* g = Graph_create(4);
  Graph_addEdge(g,0,1);
  Graph_addEdge(g,0,4);
  Graph_addEdge(g,1,3);
  Graph_addEdge(g,1,2);
  Graph_addEdge(g,2,3);
  Graph_addEdge(g,4,3);
  printf("BFS traversal: ");
  BFS(g,0);
  int count = DFS(g,0);
  if(count == g->V+1){
  printf("\n Graph is connected");
  }
  else{
   printf("\n Graph is disconnected");
  }
}
```

Output:

BFS Traversal: 0 1 4 2 3
Graph is connected

+HackerRank Problem on Tree:

functions =>

```
void inOrderTraversal(Node* root, int*
 results, int *index){
  if(root == NULL)
  return;
  inOrderTraversal(root->left, results, index
```

```
result [ ( * index ) ++ ] = root -> data;
inOrderTraversal ( root -> right, result, index);
}

void swapAtLevel ( Node * root, int k, int level) {
if (root == NULL)
return;
if ( level % k == 0)
{
Node * temp = root -> left;
root -> left = root -> right;
root -> right = temp;
}
swapAtLevel ( root -> left, k, level + 1);
swapAtLevel ( root -> right, k, level + 1);
}
```

Output:

Input (stdin)

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

Output:

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

29/02/24 Week (10):

Lab Program: Given a file of N employee records with a set of keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digits) of locations in HT.

Let the keys in K and addresses in L are integers. Design and develop a Program in C, that uses Hash function H: K → L as H(K) = K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision using linear probing.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

struct Employee {
    int key;
};
struct HashTable {
    struct Employee* table[SIZE];
};

void initializeHashTable(struct HashTable *ht
{
    for(int i=0; i<SIZE; i++)
    {
        ht->table[i]=NULL;
    }
}
```

```
int hashFunction (int key)
{
return key % SIZE;
}
void insert (struct HashTable * ht, int key,
struct Employee * emp)
{
int hKey = hashFunction (key);
int index = hKey;
int i = 0
while (ht -> table [index] != NULL)
{
i++;
index = (hKey + i) % SIZE;
}
ht -> table [index] = emp;
}
struct Employee * search (struct HashTable *ht
int key)
{
int hKey = hashFunction (key);
int index = hKey;
int i = 0;
while (ht -> table [index] != NULL) {
if (ht -> table [index] -> key == Key)
{
return ht -> table [index];
}
i++;
index = (hKey + i) % SIZE;
}
return NULL;
}
```

```
void display(struct HashTable *ht)
{
printf("\n Hash Table : \n");
for (int i=0; i<SIZE; i++)
{
 if (ht->table[i] != NULL)
 {
 printf("Index %d : Key : %d \n", i, ht->table
 [i]->key);
 }
}
}

int main()
{
 struct HashTable ht;
 initializeHashTable(&ht);

 struct Employee emp1 = {0, 01};
 struct Employee emp2 = {020, 1};
 struct Employee emp3 = {030, 1};

 insert(&ht, emp1.key, &emp1);
 insert(&ht, emp2.key, &emp2);
 insert(&ht, emp3.key, &emp3);

 display(&ht);

 int searchKey = 0201;
 struct Employee* result = search(&ht, search
 Key);
 if (result != NULL) {
 printf("\n Employee with %d found!\n",
 searchKey);
 }
```

```
else
{
printf ("\n Employee with key %d not found!
\n", searchKey);
}
return 0;
}
```

Output:

```
Hash Table:
Index 1 : Key 0101
Index 2 : Key 0201
Index 3 : Key 0301

Employee with Key 0201 found.
```