INDEX

Name _____ Std _____ Sec _____

Roll No. _____ Subject _____ School/College _____

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

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
	7/10/23	Sample program	10	7/12
	21/10/23	LAB 1	10	21/12
	28/10/23	LAB 2	10	28/12
	25/10/23	LAB 3	10	
	11/1/24	LAB 4	10	8/1/24.
	18/1/24	Lab 5	10	18/1/24
	25/1/24	Lab 6	10	20/1/24
	1/02/24	LAB 7	10	22/2/24
	15/2/24	LAB 8	10	
	21/2/24	LAB 9	10	29/2/24
		LAB 10	10	

10

29/2/24

Date / /
Page

Week-01 7/12/23

1) Enter
1 to make a new account
2 to with draw
3 to deposit
4 to check balance
5 to exit

1
Enter account name
Rajwal
Enter age 18
4
Balance is 5500
3
Enter the amount to deposit
600
2
Enter amount to with draw
1000
5

3) Enter the element to search
9

$$\begin{bmatrix} 4 & 8 & 3 \\ 7 & 5 & 9 \\ 7 & 2 & 6 \end{bmatrix}$$

9 is present

4) Enter the string
Roy name is Rajwal
Enter substring to be searched
Ry

Substring is present

5) Enter the elements
11 22 33 44 55 66 77 88 99
Enter element to search
77
last occurence of 77 is at 7

6) Enter elements
11 22 33 44 55 66 77 88 99
Enter element to be searched
22
Element 22 is at 2

7) Enter elements
11 22 33 44 55 66 77 88 99
Enter element to be searched
44
Element 44 is at 4

8) Enter number of element
5
11 22 33 44 55
Biggest is 55
Smallest is 11

LAB - 01

Date __/__/__
Page _____

1) Swapping of two numbers using Pointers

```
#include <stdio.h>
int main()
{
    int a, b, temp;
    int *ptr1, *ptr2;
    printf("Enter the value of a and b");
    scanf("%d %d", &a, &b);
    printf("\n Before swapping a=%d and
    b=%d", a, b);
    ptr1 = &a;
    ptr2 = &b;
    temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
    printf("\n After Swapping a=%d and
    b=%d", a, b);
    return 0;
}
```

Output

Enter the value of a and b
20
30
Before swapping a=20 and b=30
After swapping a=30 and b=20

Date / /
Page

Q) Stack Implementation

```
#include<stdio.h>
int stack[100], choice, n, top, x, i;
void push(void);
void pop(void);
void display(void);
int main()
{
    top = -1;
    printf("\n Enter the size of STACK
        [MAX=100]: ");
    scanf("%d", &n);
    printf("\n\t Stack Operations Using Array");
    printf("\n\t ----------");
    printf("\n\t 1. Push 2. Pop\n\t
     3. Display\n\t 4. EXIT");
    do
    {
      printf("\n Enter the choice: ");
      scanf("%d", &choice);
      switch(choice)
      {
        case 1:
        {
          push();
          break;
        }
        case 2:
        {
          pop();
          break;
        }
```

```
case 3:
{
    display ();
    break;
}
case 4:
{
    printf ("\n\t Exit point ");
    break;
}
default:
{
    printf ("\n\t Please Enter a Valid
        Choice (1/2/3/4)");
}
}
}
while (choice != 4);
return 0;
}
void push()
{
    if (top >= SIZE-1)
    {
        printf ("\n\t STACK is over flow");
    }
    else
    {
        printf ("Enter a value to be pushed:");
        scanf ("%d", &x);
        top++;
        stack [top] = x;
    }
```

```
}
void pop()
{
    if (top <= -1)
    {
        printf ("\n\t stack is under flow");
    }
    else
    {
        printf ("\n\t The popped element is
        %d", stack [top]);
        top--;
    }
}
void display()
{
    if (top >=0)
    {
        printf ("\n The elements in STACK\n");
        for (i=top; i>=0; i--)
        printf ("\n%d", stack [i]);
        printf ("\n Press Next Choice");
    }
    else
    {
        printf ("\n The STACK is empty");
    }
}
```

Output

Enter the size of STACK : 10

STACK Operation Using Array
1. PUSH
2. POP
3. Display
4. Exit

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

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

Enter the choice : 3
~~The~~ Enter a value to be pushed : 98

Enter the choice : 3

The elements in STACK
98
24
12

Press Next choice.
Enter the choice : 2
The popped element is 98
Enter the choice : 3
The elements in STACK
24
12

LAB-02

08/12/23

1) Infix to Postfix

```
#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;
}
```

Date
Page

Output

Enter infix expression:
A^B+C^D-E
Postfix expression:
AB^CD^+E-

2) Postfix Evaluation

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

#define MAX 100
int stack[MAX];
int top = -1;

void push(int item) {
    if (top >= MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = item;
}

int pop() {
    if (top < 0) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack[top--];
}
```

```
int evaluate_postfix (char postfix[]) {
    int i, operand1, operand2, result;
    char ch;
    for (i=0; postfix[i] != '\0'; i++) {
        ch = postfix[i];
        if (isdigit(ch)) {
            push (ch - '0');
        } else {
            operand2 = pop();
            operand1 = pop();
            switch (ch) {
                case '+':
                    result = operand1 + operand2;
                    break;
                case '-':
                    result = operand1 - operand2; break;
                case '*':
                    result = operand1 * operand2;
                    break;
                case '/':
                    result = operand1 / operand2;
                    break;
                default:
                    printf ("Invalid Expression\n");
                    exit (1);
            }
            push (result);
        }
    }
    return pop();

int main() {
    char postfix[MAX];
```

```
printf("Enter postfix expression : ");
gets(postfix);
int result = evaluatePostfix(postfix);
printf("Result : %d \n", result);
return 0;
}
```

Output

Enter postfix expression : 12^34^+5-
Result : 9

date
sign

Date __/__/__
Page _____

LAB - 4

Pgm - 3b

```
#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 dequeued element 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;
        }
```

Date __/__/__
Page _____

```
    }
}
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;
}
```

Date / /
Page

Output

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 10

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice 2

The dequeued element is 10

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice 3
Queue is empty

3 a)

```
#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 \n1. Insert \n2.
Delete \n3. Display \n4. Exit \n");
scanf("%d", &option);
switch(option)
{
case 1: enqueue();
break;
case 2: val = dequeue();
printf("Element delete 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 ("Element 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. ~~Insert~~ Delete
3. Display
4. Exit
Enter the number to be inserted in the queue
18

Enter an option to Perform operation
1. Insert
2. Delete
3. Display
4. Exit
Element deleted from queue is 18
Enter an option to perform following operation
1. Insert
2. Delete
3. Display
4. Exit
3
Element in the queue;
Underflow

Lab - 4

Pgm 4

```
#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 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;
    printf("Enter data in the new node: ");
    scanf("%d", &new_data);
    temp->data = new_data;
    temp->next = NULL;
        head = temp;
        return;
```

```
}
Node *temp1 = head;
while (temp1->next != NULL) {
    temp1 = temp1->next;
}
temp1->next = temp;
}

Void insert() {
    Node *temp = (Node*) malloc (sizeof(Node));
    int new_data, pos;
    printf ("Enter data in new node: ");
    scanf ("%d", &new_data);
    printf ("Enter position of new node: ");
    scanf ("%d", &pos);
    temp->data = new_data;
    temp->next = NULL;
    if (pos == 0) {
        temp->next = head;
        head = temp;
        return;
    }
    Node *temp1 = head;
    while (temp1->next != NULL) {
        temp1 = temp1->next;
    }
    temp1->next = temp;
}

Void insert() {
    Node *temp = (Node*) malloc (size of
        (Node));
```

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

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

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 data node : 17
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice : 2
Enter data in the new node : 19
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice : 2
Enter data in new node : 10
1. Insert at beginning
2. Insert at position end
3. Insert at position
4. Display
5. Exit

Date __/__/__
Page

Roster chain 4

17 -> 19 -> 10 -> NULL

Week 4 Lab Programs

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

struct node {
 int info;
 struct node *link;
};
struct node *start =NULL;

void createList()
{
  if (start == NULL) {
    int n;
    printf("\n Enter the number of
     nodes:");
    scanf("%d", &n);

    if (n>0){
      int data;
      struct node *newnode;
      struct node *temp;

    newnode = malloc(sizeof(struct node));
    start =newnode;
    temp =start;

    printf("\n Enter number to be
       inserted :");
    scanf("%d", &data);
    start -> info =data;
```

```
for (int i = 0; i <= n; i++) {
    new node = malloc (sizeof (struct node));
    temp -> link = new node;

    printf ("\n Enter number to be inserted :");
    scanf ("%d", &data);
    new node -> info = data;
    temp = temp -> link;
  }
    temp -> link = NULL;
}
  printf ("\n The list is created \n");
} else {
  Printf ("\n The list is already
    created \n");
  }
}

void display ()
{
struct node * temp;
if (start == NULL)
printf ("\n List is empty \n");
  else {
temp = start;
while (temp != NULL) {
printf ("Data = %d \n", temp -> info);
temp = temp -> link;
}
}
}
```

```
void deleteFirst()
{
struct node *temp;
if(start == NULL)
printf("\n List is empty \n");
  else{
  temp = start;
  start = start -> link;
  free(temp);
  }
}

void deleteEnd()
{
struct node *temp, *prevnode;
if (start == NULL)
printf("\n List is Empty \n");
else{
temp = start;
while (temp -> link != NULL){
prevnode = temp;
temp = temp -> link;
}
free(temp);
prevnode -> link = NULL;
}
}

void deletePosition()
{
  struct node *temp, *position, *prevnode;
  int i=1, pos;
```

```
if (start == NULL)
    printf("\n List is empty \n");
else {
    printf ("\n Enter index : ");
    scanf ("%d", &pos);

    if (pos <= 0) {
        printf ("\n Invalid position \n");
        return;
    }

    temp = start;
    position = NULL;

    if (pos == 1) {
        start = start -> link;
        free (temp);
        return;
    }

    while (i < pos && temp != NULL) {
        precnode = temp;
        temp = temp -> link;
        i++;
    }

    if (temp == NULL) {
        printf ("\n Invalid position \n");
        return;
    }

    position = temp;
    precnode -> link = temp -> link;
```

```
        free(position);
    }
}

int main()
{
 create List();
 int choice;
 while (1) {
 printf("\n\t1. To see List\n");
 printf("\t2. For deletion of" first
   element\n");
 printf("\t3. For deletion of" last
      element\n");
 printf("\t4. For deletion of "element
   at any position\n");
 printf("\n Enter choice :\n");
  scanf("%d", &choice);

 switch (choice) {
 case 1:
 display();
  break;
 case 2:
 deleteFirst();
   break;
 case 3:
 deleteEnd();
  break;
 case 4:
 deletePosition();
 break;
```

```
case 5:
exit (1);
break;
default:
printf("Incorrect choice\n");
}
}
return 0;
}
```

Output

Enter the number of nodes: 3

Enter number to be inserted: 1
Enter number to be inserted: 2
Enter number to be inserted: 3

The List is created
1. To see list
2. For deletion of first element
3. For deletion of last element
4. For deletion of element at any position
5. To exit

Enter choice:
2

1. To see list
2. For deletion of first element
3. For deletion of last element
4. For deletion of element at any position
5. To exit

Enter choice:
1

Data = 2
Data = 3

Enter choice:
3

1. To see list
2. For deletion of First element
3. For deletion of Last element
4. For deletion of element at any position
5. To exit

Enter choice:
1

Data = 2

Leet code - 1

```
#include <stdlib.h>

typedef struct {
int* stack;
int* minStack;
int top;
} MinStack;

MinStack* minStackCreate() {
MinStack* stack = (MinStack*) malloc
(sizeof(MinStack));
```

```
stack -> stack = (int*) malloc (sizeof
  (int) * 50);
stack -> minstack = (int*) malloc (sizeof
  (int) * 50);
stack -> top = -1;
return stack;
}

void minstackPush (minstack * obj,
  int val){
obj -> top++;
  obj -> stack [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 minstackPop (minstack * obj ){
  obj -> top--;
}

int minstackTop (minstack * obj){
    return obj -> stack [obj -> top];
}

int minstackGetmin (minstack * obj){
    return obj -> minstack [obj -> top];
}
```

Date __/__/__
Page

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

Output

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

Expected

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

18/1/24

Leet code 2

Date __/__/__
Page _____

```
struct ListNode* reverseBetween
        (struct ListNode* start, int a, int b)
{
    a--;
    b--;
    struct ListNode *node1 = NULL,
        *node2 = NULL, *nodeb = NULL;
        *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++;
    ptr = ptr->next;
}
struct ListNode* ptr = node, *temp;
    ptr = start;
    c=0;
while (ptr != NULL)
    {
```

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

Output

head

[1, 2, 3, 4, 5]

left = 2

height = 4

Output

[1, 4, 3, 2, 5]

LAB-6

Date __/__/__
Page

```
#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 *** MAIN MENU ***\n
\n1. Create a linked list \n2. Create
two linked lists for concatenation
\n3. Sort \n4. Concatenate \n5. Reverse
\n6. Display linked list \n7. Display
Concatenated linked list \n8. Exit
\n");
scanf ("
printf ("\n Enter an option to perform
the following operations: ");
```

```
scanf ("%d", &option);
switch (option)
{
  case 1: start = create (start);
          printf ("\n Linked list created
          succesfully \n");
          break;
  case 2: printf ("\n Linked list 1: \n");
          s1 = creat (s1);
          printf ("\n Linked list 2: \n");
          s2 = creat (s2);
          printf ("\n Linked list created
          succesfully \n");
          break;
  case 3: sort ();
          printf ("\n Linked list sorted \n");
          break;
  case 4: a = concatenate (s1, s2);
          printf ("\n Linked list concatenated
          succesfully \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");
          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;
```

1

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

Output

**** MAIN MENU ****

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

Enter an option to perform the following operations: 1

Enter -1 to exit

Enter the data : 2

Enter the data : 1

Enter the data : -1

Linked list created successfully

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

Enter an option to perform the following operations: 5

Linked list reversed

Enter an option to perform following
Operations: 6
Elements in the linked list
1 2

```
#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 \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;
        }
    }
    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 operation

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

1

Enter the data

2

Enter the number to perform following operation

1. Push
2. Pop

3. Display
4. Exit
2
Element popped from the stack
is: 8

b) #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 the number to perform following operation \n1. Enqueue \n2. Dequeue \n3. Display \n4. 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, *prept;
ptr = start;
if (start == NULL)
{
printf ("stack is empty \n");
exit (0);
}
else if (start -> next == NULL)
{
start = start -> next;
printf ("\n Element popped from
the stack is: %d \n", ptr -> data);
```

```
        free(ptr);
    }
    else
    {
    while(ptr->next != NULL)
    {
    preptr = ptr;
    ptr = ptr->next;
    }
    preptr->next = NULL;
    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");
}
```

Date __/__/__
Page _____

Output

Enter the number to perform following operations
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
4. Exit
2

Element popped from the stack is : 10

28/11/24

LAB-7

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

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

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

void insertLeft(struct Node** head, int
value, int target) {
struct Node* newNode = CreateNode
(value);
struct Node* current = *head;

while (current != NULL && current->
data != target) {
    current = current->next;
}

if (current != NULL) {
    newNode->prev = current->prev;
```

```
    newNode->next = current;
    if (current->prev != NULL) {
        current->prev->next = newNode;
    } else {
        *head = newNode;
    }
    current->prev = newNode;
} else {
    printf("Node with value %d not found.\n", target);
}
}

void deleteNode(struct Node** head, int value) {
    struct Node* current = *head;

while (current != NULL && current->data != value) {
    current = current->next;
}

if (current != NULL) {
    if (current->prev != NULL) {
        current->prev->next =
        current->next;
    } else {
        *head = current->next;
    }
    if (current->next != NULL) {
        current->next->prev =
        current->prev;
}
```

```
        free(current);
    } else {
        printf("Node with value %d
        not found.\n", value);
    }
}

void displayList(struct Node* head){
    struct Node* current = head;

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

int main(){
    struct Node* head = NULL;
    int choice, value, target;

    do{
        printf("\n Menu : \n");
        printf("a) Create a double linked
        list \n");
        printf("2) Insert a new node to the
        left of the node \n");
        printf("3) Delete the node based
        on a specific value \n");
        printf("4) Exit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
```

```
switch (choice) {
    case 1:
        if (head != NULL) {
            printf("Doubly linked list
            already exists\n");
        } else {
            head = createNode(1);
            head -> next = createNode(2);
            head -> next -> prev = head;
            head -> next -> next =
            createNode(3);
            head -> next -> next -> prev
            = head -> next;
            printf("Doubly linked list created
            \n");
        }
        break;

    case 2:
        printf("Enter the value to be inserted: ");
        scanf("%d", &value);
        printf("Enter the target value: ");
        scanf("%d", &target);
        insertLeft(&head, value, target);
        printf("After insertion: ");
        displayList(head);
        break;

    case 3:
        printf("Enter the value to be deleted:
        scanf("%d", &value);
        deleteNode(&head, value);
        printf("After deletion: ");
```

```
        display list (head);
        break;

    case 4:
        printf ("Exiting the program \n");
        break;

    default:
        printf ("Invalid choice. Please
        enter a valid option. \n");
    }
} while (choice != '4');
    return 0;
}
```

Output

```
Menu:
1) Create a double linked list
2) Insert a new node to the left of
   the node
3) Delete the node based on a
   specific value
4) Exit

Enter your choice: 1
Doubly linked list created

Menu:
1) Create a doubly linked list
2) Insert a new node to the left
   of the node
```

c) Delete the node based on a specific value
d) Exit
Enter your choice: 2
Enter the value to be inserted: 2
Enter the target value: 3
After insertion: 1 -> 2 -> 2 -> 3 -> NULL

Enter your choice: 2
Enter the value to be inserted: 6
Enter the target value: 3
After insertion: 1 -> 2 -> 2 -> 6 -> 3 -> NULL

Menu:
a) Create a doubly linked list
2) Insert a new node to the left of node
3) Delete the node based on specific value
4) Exit
Enter your choice: 3
Enter the value to be deleted: 2
After deletion: 1 -> 0 -> 6 -> 3 -> NULL

* 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 segment_size = num + (a-- > 0);
    for(int j=1; j<segment_size; j++){
        ptr = ptr->next;
    }
    if(ptr != NULL){
    struct ListNode* next = ptr->next;
        ptr->next = NULL;
        ptr = next;
    }
}
return L;
}
```

Output

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

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

Lab 8

Date __/__/__
Page _____

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

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

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

struct node *insert (struct node *root,
 int data) {
 if (root == NULL) {
    return newNode (data);
 }

struct node *insert (struct node *root,
 int data) {
 if (root == NULL) {
    return newNode (data);
 }

else {
```

```
    if (data < root -> data) {
    root -> left = insert (root -> left, data);
    }
    else if (data > root -> data) {
    root -> right = insert (root -> right,
    data);
    }
}
return root;
}

void inOrder (struct node *root) {
    if (root != NULL) {
    inOrder (root -> left);
    printf ("%d", root -> data);
    inOrder (root -> right);
    }
}

void preOrder (struct node *root)
{
    if (root != NULL) {
    printf ("%d", root -> data);
    preOrder (root -> left);
    preOrder (root -> right);
    }
}

void postOrder (struct node *root)
{
    if (root != NULL) {
    postorder (root -> left);
    postorder (root -> right);
```

```
        printf("%d", root->data);
    }
}

void display(struct node *root){
    printf("In-order traversal: ");
    inorder(root);
    printf("\n");
    printf("Pre-order traversal: ");
    preorder(root);
    printf("\n");
    printf("Post-order traversal: ");
    postorder(root);
    printf("\n");
}

int main(){
    struct node *root = NULL;
    int data;
    printf("Enter the elements of the
        tree (-1 to stop): ");
while(scanf("%d", &data) && (
    data != -1){
    root = insert(root, data);
}
display(root);
free(root);
return 0;
}
```

Date
Page

Output

(-1 to stop)

Enter the elements of the tree : 7
9
8
2
3
4
1
-1

In order traversal : 1 2 3 4 7 8 9
Pre-order traversal : 7 2 1 3 4 9 8
Post-order traversal : 1 4 3 2 8 9 7

Leetcode-4

```
class Solution{
public:
  ListNode* reverse(ListNode* head){
    ListNode *curr = head, *prev = NULL,
    *forward = NULL;
    while(curr != NULL){
      forward = curr->next;
      curr->next = prev;
      prev = curr;
      curr = forward;
    }
    return prev;
  }
int getlength(ListNode* head){
    int len = 0;
    while(head != NULL){
```

```
        head = head -> next;
            len ++;
        }
        return len; }
    List node * rotateRight (List Node * head, int k){
        if (head == NULL || head -> next == NULL)
            return head;
        int len = getLength (head);
            k = k % len;
            if (k == 0)
                return head;
        List Node * newHead = reverse (head);
        List Node * p1 = newHead, * p2 = newHead,
        * prev1 = newHead;
        while (k--){
                prev1 = p1;
                p1 = p1 -> next;
            }
    prev1 -> next = NULL;
        List Node * finalHead = reverse (p2);
        List Node * join Head = reverse (p1);
        p1 = finalHead;
        while (p1 -> next != NULL)
                p1 = p1 -> next;
        p1 -> next = join Head;
            return finalHead;
        }
    };
```

Output

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

k = 2

Output: {4, 5, 1, 2, 3}
Expected: {4, 5, 1, 2, 3}

Lab: 9 BFS

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct Queue
{ int item[MAX_SIZE];
  int front, rear; };
struct Queue* createQueue()
{ struct Queue* queue = (struct Queue*) malloc
(sizeof(struct Queue));
  queue->front = -1;
  queue->rear = -1;
  return queue;
}
int isEmpty(struct Queue* queue)
{
  if (queue->rear == -1)
    return 1;
  else return 0;
}
void enqueue(struct Queue* queue, int val)
{ if (queue->rear == MAX_SIZE-1)
  printf("\n Queue is FULL!!");
else {
  if (queue->front == -1)
    queue->front = 0;
  queue->rear++;
```

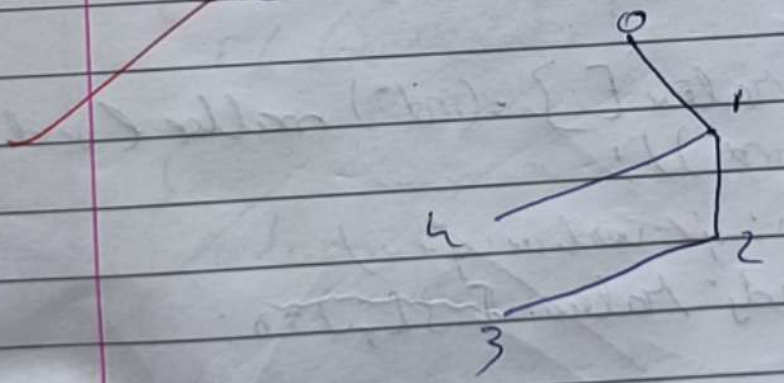
```
queue->item[queue->rear]=value; } }
int dequeue(struct Queue* queue){
int item;
if(isEmpty(queue)){
printf("Queue is Empty");
item=-1; }
else{
item=queue->item[queue->front];
queue->front++;
if(queue->front->queue->rear)
{ queue->front=queue->rear=-1;
}
}
return item;
}
struct Graph{
int vertices;
int** adjMatrix; };
struct Graph* creatGraph(int vertices){
struct Graph* graph=(struct Graph*)malloc
(sizeof(struct Graph));
graph->vertices=vertices;
graph->adjMatrix=(int**)malloc(vertices
*sizeof(int*));
for(int i=0;i<vertices;i++){
graph->adjMatrix[i]=(int*)malloc
(vertices*sizeof(int));
for(int j=0;j<vertices;j++)
graph->adjMatrix[i][j]=0;}
return graph;}
void addEdge(struct Graph* graph,int src,
int dest){
graph->adjMatrix[src][dest]=1;
```

```
graph->adjMatrix[dest][src] = 1; }
void BFS(struct Graph* graph, int startVertex)
{ int visited[MAX_SIZE] = {0};
struct Queue* queue = createQueue();
visited[startVertex] = 1;
enqueue(queue, startVertex);
printf("Breadth First Search Traversal: ");
while (!isEmpty(queue)) {
int currentVertex = dequeue(queue);
printf("%d ", currentVertex);
for (int i=0; i<graph->vertex; i++)
printf("%d", currentVertex);
for(int i=0; i<graph->vertices; i++)
{
if(graph->adjMatrix[currentVertex][i]==1
visited[i] == 0) {
visited[i] = 1;
enqueue(queue, i);
}
}
{ printf("\n"); }
int main()
{
int vertices, edges, src, dest;
printf("Enter the no of vertices: ");
scanf("%d", &vertex);
struct Graph* graph = createGraph(vertices);
printf("Enter the no of edges: ");
scanf("%d", &edges);
for(int i=0; i<edges; i++)
{
printf("Enter edge %d (source dest): ",
i+1);
```

```
scanf("%d %d", &src, &dest);
addEdge(graph, src, dest);
}
int startVertex;
Printf("Enter the starting vertex for BFS: ");
Scanf("%d", &startVertex);
BFS(graph, startVertex)
return 0;
}
```

Output

Enter the number of vertices: 5
Enter the no of edges: 4
Enter edge 1 (src, dest): 01
Enter edge 2 (src, dest): 03
Enter edge 3 (src, dest): 14
Enter edge 4 (src, dest): 12
Enter the starting vertex for BFS: 0
Breadth Search Traversal: 0 1 2 4 3



Date / /
Page

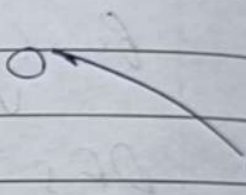
DFS

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

struct Graph
{
    int vertices;
    int ** adjMatrix;
};

struct Graph* createGraph(int vertices)
{
struct Graph* graph = (struct Graph *)
malloc (size of (struct Graph));
  graph->vertices = vertices;
  graph->adjMatrix = (int**) malloc(vertices
   * size of (int));
  for (int i=0; i<vertices; i++)
  {
  graph->adjMatrix[i] = (int*) malloc (vertices
   * size of (int));
 for (int j=0; j<vertices; j++)
    graph->adjMatrix[i][j] = 0;
  }
  return graph;
}
void addEdge (struct Graph* graph, int src, int dest)
{
  graph->adjMatrix[src][dest] = 1;
  graph->adjMatrix[dest][src] = 1;
}
```

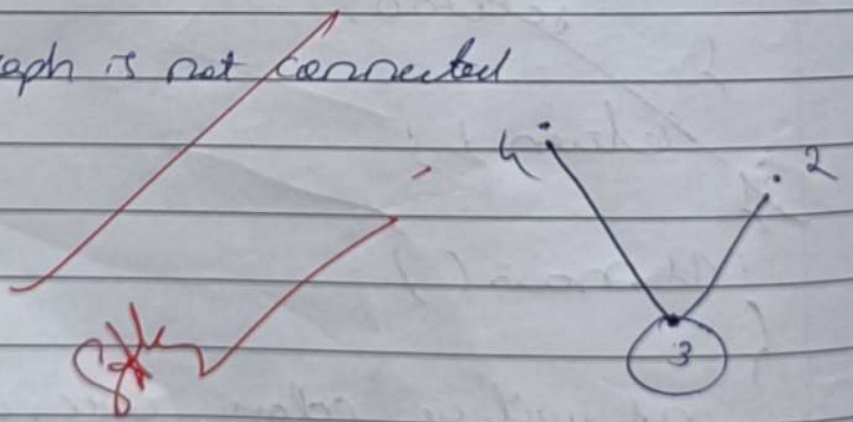
```
for (int i=0; i< edges; i++)
{
printf ("Enter edge -%d (src dest): ", i+1);
scanf ("%d %d", &src, &dest);
addEdge (graph, src, dest);
}
if (isConnected (graph))
printf ("The graph is connected \n");
else
printf ("The graph is not connected \n");
return 0;
}
```

Output

Enter the no of vertices: 5
Enter the no of edges: 4
Enter edge 1 (src dest) = 0 1
Enter edge 2 (src dest) = 0 3
Enter edge 3 (src dest) = 3 3
Enter edge 4 (src dest) = 3 4



The graph is not connected



Date Page

Hackerrank

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

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

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

void inOrderTraversal (Node *root, int *result,
    int *index) {
    if (root == NULL) return;
    inOrderTraversal (root -> left, result,
    index);
    result [(*index)++] = root -> data;
    inOrderTraversal (root -> right, result, index);
}

void sweepAtLevel (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;
}
int** sweepNodes(int indexes_rows, int indexes_columns, int** indexes, int queries_count,
int* queries, int* result_rows, int* result_columns){
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 leftIndex = indexes[i][0];
int rightIndex = 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_count*sizeof(int*));
*result_rows = queries_count;
*result_columns = indexes_rows;
for(int i=0; i<queries_count; i++){
sweepAtLevel(nodes[1], queries[i], 1);
int* traversedResult = (int*)malloc(indexes_rows*sizeof(int));
int index=0;
inOrderTraversal(nodes[1], traversedResult,
```

```
&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_count;
scanf("%d", &queries_count);
int* queries = malloc(queries_count * sizeof
(int));
for(int i=0; i<queries_count; i++){
  scanf("%d", &queries[i]);
}
int result_rows;
int result_columns;
int** result = treeQueries(n, 2, indexes, queries_
count, 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);
for(int i=0; i<n; i++){
free(indexes[i]);
}
free(indexes);
free(queries);
*returnSize=0;
}
```

Output

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

→

Your O/P
3 1 2
2 1 3

Expected O/P
3 1 2
2 1 3

29/2/24

Lab 10 Hashing

```
#include <stdio.h>
#define TABLE_SIZE 10

int hashFunction(int key){
        return key % TABLE_SIZE;
}

void insertValue(int hashTable[], int key)
{
    int i=0;
    int hKey = hashFunction(key);
    int index;

    do{
        index = (hKey + i) % TABLE_SIZE;
        if (hashTable[index] == -1){
            hashTable[index] = key
        printf("Inserted key %d at index %d\n",
        key, index);
        return;
        }
        i++
    }while (i<TABLE_SIZE);
    printf("unable to insert key %d. Hash Table
    is full\n", key);
}
int searchValue(int hashTable[], int key)
{
    int i=0;
    int key = hashFunction(key);
    int index;
```

```
do{
    index = (hkey + i) % TABLE_SIZE;
    if (hashTable[index] == key){
    printf("Key %d: d found at index %d\n",
    key, index);
    return index;
    }
    i++;
}while (i < TABLE_SIZE);
printf("Key %d not found in hash Table\n",
    key);
    return -1;
}

int main(){
    int hashTable[TABLE_SIZE];
    for (int i=0; i< TABLE_SIZE; i++){
        hashTable[i] = -1;
    }
    insert value (hashTable, 8);
    insert value (hashTable, 46);
    insert value (hashTable, 7);
    insert value (hashTable, 17);
    insert value (hashTable, 1);

    return 0;
}
```

Output: inserted key 8 at index 8
inserted key 46 at index 6
inserted key 7 at index 7
inserted key 17 at index 9
inserted key 1 at index 2

Rachner Rank:

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

struct node
{
    int id;
    int depth;
    struct node * left, *right;
};

void
norder (struct node * tree)
{
    if (tree == NULL)
        return;
    norder (tree -> left);
    printf ("%d", tree -> id);
    norder ((tree -> right));
}

int main (void)
{
    int no_of_node, i=0;
    int l, r, max_depth=0;
    struct node * temp = NULL;
    scanf ("%d", &no_of_node);
struct node * tree = (struct node *) calloc
    (no_of_node, sizeof (struct node));
    tree[0] depth = 1;
    while (i < no_of_node)
    {
```

```
tree[i].id = i+1;
scanf("%d %d %d", &i, &l, &r);
if (l == -1)
    tree[i].left = NULL;
else
{
    tree[i].left = &tree[l-1];
    tree[i].left->depth = tree[i].depth + 1;
    max_depth = tree[i].left->depth;
}

if (r == -1)
    tree[i].right = NULL;
else{
    tree[i].right = &tree[r-1];
    tree[i].right->depth = tree[i].depth+1;
    max_depth = tree[i].right->depth+1;
}

i++
{
scanf("%d", &i);
while(i--)
{
scanf("%d", &l);
    r=l;
while(l<=max_depth)
    {
for(k=0; k<no_of_nodes; ++k)
if (tree[k].depth == -1)
    {
tree = tree[k].left;
```

```
tree[k].left = tree[k].right;
  tree[k].right = temp;
  }
 }
 l = l+r;
 {
  inorder(tree);
   printf("\n");
 }
  return 0;
}
```

Output

```
Enter Key: 5
Enter Data: 50

Enter Key: 15
Enter Data: 150

Enter Key: 25
Enter Data: 250

Enter Key: 35
Enter data: 350
Hash Table:
Index 0: Empty
Index 1: Empty
Index 2: Empty
Index 3: Empty
Index 4: Empty
Index 5: Empty
Index 6: Key 5, Data 50
Index 7: Key 15, Data 150
Index 8: Key 25, Data 250
Index 9: Key 35, Data 350
Index 10: Empty
```