INDEX

Name P. Manya Std _____ Sec 3-D

Roll No. _____ Subject DS Lab School/College _____

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

8 practice programs outputs:

1. Enter your name
manya:p
Enter account number
98765
Enter amount
100000000
Enter amount to be withdrawed
34567880
Remaining balance = 96543220
Enter amount to be deposited
10000
Balance = 96553220

2. Enter 5 words: END
BORING
MANYA
NECKlace
hungry
In lexographical order:
BORing
MANYA
NECKlace
END
hungry

3. Enter the no of elements in the array: 3
Enter the elements: 111
222
333
Enter no to be searched: 222
222 is found at position = 2

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

5. Enter size of array: 5
Enter the elements of the array:
Element 1: 4
Element 2: 7
Element 3: 2
Element 4: 7
Element 5: 9
Enter the no to find the last occurence
The last occurence of 7 is at index 3

6. Enter no of elements in array: 5
Enter elements: 890
876
567
867
874
Enter no to be searched: 867
867 is found at position = 4

7. Enter the no of element in array: 7
Enter the elements: 12
13
23
24
35
36
47
48
59

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

8. Enter size of the array : 3
Enter elements in the array : 12345678
67344548 5.
98553773 8
Maximum element = 98553773 8
Minimum element = 1234578

Date:

LAB 2

Pgm-1

1.) Swapping of 2 nos using pointers

```
#include <stdio.h>
void swap (int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main()
{
    int num1, num2;
printf(" Enter first number : ");
scanf(" %d", &num1);
printf(" Enter second no ");
scanf(" " before swapping : num1 = %d,
    \n", num1, num2);
swap (&num1, &num2);
printf(" After swapping : num1 = %d,
    num2 = %d \n", num1, num2)
return 0;
}
```

Output:

Enter 1st number : 12

Enter second number : 24

Before swapping : num1 = 12, num2 = 24

After swapping : num1 = 24, num2 = 12

Date: / /

2. Pgm - 2

```
#include <stdio.h>
#include <stdlib.h>
void *mymalloc (size_t size)
{
return malloc (size);
}
void realloc (void * ptr, size_t size)
{
return realloc (ptr, size);
}
void *calloc (size_t num, size_t size)
{
return calloc (num, size);
}
void free (void *ptr)
{
free (ptr);
}
int main()
{
int *arr1, *arr2;
size_t size;
printf(" Enter size");
scanf("%d", &size);
arr1 = (int *) malloc (size * sizeof(int));
if (arr1 == NULL)
{
printf(" Memory allocation failed");
return 1;
}
printf(" Enter elements");
for (size_t i=0; i<size; i++)
{
printf(" Elements %d\n", i+1);
scanf("%d", &arr1[i]);
}
printf(" Elements of realloc");
for (size_t i=0; i< size; i++)
{
```

```
printf("%d", arr1[i]);
}
printf("\n");
size_t x = 2;
arr2 = (int*)realloc(arr1, size * sizeof(int));
if (arr2 == NULL) {
printf("Memory allocation failed.");
free(arr1);
return 1;
}
printf("Enter additional elements:\n");
for (size_t i = size; i < size; i++) {
printf("Element %d: ", i+1);
scanf("%d", &arr2[i]);
}
printf("Elements of array (realloc):\n");
for (size_t i = 0; i < size; i++) {
printf("%d", arr2[i]);
}
printf("\n");
free(arr2);
return 0;
}
```

Output

Enter size of array: 2
Enter elements of array:
Enter 1: 12
Element 2: 24
Elements of array (malloc):
12 24
Enter additional elements of array:
Element 3: 36

Element 4: 90
Elements of array (realloc):
12 24 36 90

3. Stack implementation

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

```
if (isEmpty(stack)) {
    printf("Stack is empty");
}
else {
printf("Elements in stack");
for (int i=0; i<=stack->top; i++)
{ printf("%d", stack->items[i]);
}
printf("\n");
}}
int main()
{
    struct stack stack;
    initialize(&stack);
    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    display(&stack);
    pop(&stack);
    display(&stack);
    push(&stack, 40);
    display(&stack);
    return 0;
}
```

output

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

Pushed 40 onto stack
Elements in stack: 10 20 40

Sneha
21/12/23

28/12/23 WEEK-2

1.

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50
char stack [SIZE];
int top = -1;
push (char de)
{
  stack [++ top] = de;
}
char pop()
{
  return (stack [top --]);
}
int pr ( char symbol)
{
  if (symbol == '^')
  {
    return (3);
  }
  else if (symbol == '*' || symbol == '/')
  {
    return (2);
  }
  else if ( symbol == '+' || symbol == '-')
  {
    return (1);
  }
  else
  {
    return(0); } }
```

```
void main()
{
char infix[50], postfix[50], ch, ele;
int i=0, k=0;
printf("enter infix expression");
scanf("%s", infix);
push('#');
while((ch=infix[i++])!='\0')
{
if(ch=='(') push(ch);
else
if(isalnum(ch)) postfix[k++]=ch;
else
if(ch==')')
{
while(stack[top]!='(')
postfix[k++]=pop();
ele=pop();
}
else
{
while(pro(stack[top])>=pro(ch))
postfix[k++]=pop();
push(ch);
}
}
while(stack[top]!='#')
postfix[k++]=pop();
postfix[k]='\0';
printf("\n Postfix exp=%s\n", postfix);
}
```

Date: / /

Output:

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

2. Evaluation of Postfix

```
#include <stdio.h>
#include <ctype.h>
int stack [20];
int top = -1;
void push (int x)
{
  stack [++top] = x;
}
int pop ()
{
  return stack [top--];
}
int main()
{
  char exp[20];
  char *e;
  int n1, n2, n3, num;
  printf("Enter the expression: ");
  scanf("%s", exp);
  e = exp;
  while(*e != '\0')
  {
    if(isdigit(*e))
    num = *e - 48;
```

```
push(num);
}
else
{
n1 = pop();
n2 = pop();
switch(*e)
{
case '+';
{
n3 = n1+n2;
break;
}
case '-';
{
n3 = n1-n2;
break;
}
case '*';
{
n3 = n1*n2;
break;
}
case '/';
{
n3 = n1/n2;
break;
}
push(n3);
e++;
}
printf("result of expression is %d", pop());
```

Output:
Enter the expression: 45*9+
Result of expression is 45*9+ = 29

LAB - 3

3.a.

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

```
insert();
break;
case 2:
delete();
break;
case 3:
display();
break;
printf("invalid character");
}}
insert()
{
int add_item;
if (rear == MAX-1)
printf("queue overflow");
else
{
if (front == -1)
front = 0;
printf("insert the element");
scanf("%d", &add_item);
rear = rear+1;
queue_array[rear] = add_item;
}}
delete()
{
if (front == -1 || front > rear)
{
printf("queue underflow");
return;
}
else
{
```

printf("deleted element is %d\n", queue[front]);

front += 1;

}

Output

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

enter your choice: 1

insert the element in queue: 3

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

enter your choice: 1

insert element in queue: 3

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

enter your choice: 2

deleted element is: 3

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

enter your choice: 3

queue is:

39

3.b.

Sneha 28/12/23

Date: / /

3.b. Circular Queue

```
#include <stdio.h>
#define SIZE 5
int items[SIZE];
int front = -1, rear = -1;
int isFull() {
  if ((front == rear + 1) || (front == 0 && rear
  == size - 1))
    return 1;
  return 0;
}
int isEmpty() {
  if (front == -1)
    return 1;
  return 0;
}
void enqueue(int element) {
  if (isFull)
    printf("\n Queue is full");
  else {
    if (front == -1)
      front = 0;
    rear = (rear + 1) % SIZE;
    items[rear] = element;
    printf("\n Inserted = %d", element);
  }
}
int dequeue() {
  int element;
  if (isEmpty()) {
    printf("\n Queue is empty");
    return -1;
  }
```

```
else {
 element = items [front];
 if (front == rear)
 {
   front = -1;
   rear = -1;
   }
   else
   {
   front = (front + 1) % SIZE;
   }
  printf(" Delete element -> %d \n")
   return element;
   }
  }
void display () {
   int i;
  if (is Empty())
   printf(" Empty Queue ");
  else {
  printf(" Front -> %d", front);
  printf(" items ->");
 for (i = front; i != rear; i = (i+1) % SIZE) {
  printf("%d", items [i]);
 }
 printf("%d", items [i]);
 printf(" Rear -> %d \n", rear);
 }
 }
int main () {
 enQueue (1);
 enQueue (2);
 enQueue (3);
```

```
enQueue(4);
enQueue(5);
display();
deQueue();
deQueue();
display();
enQueue(6);
enQueue(7);
display();
return 0;
}
```

Output:

Inserted 1
Inserted 2
Inserted 3
Inserted 4
Inserted 5
Front - 0
Items - 12345
Rear 4

Deleted element 1
Deleted element 2

Front 2
Items 345
Rear 4
Inserted 6
Inserted 7
Front 2
Items 3 4 5 6 7
Rear 6

11/1/24 LAB-4 SINGLY LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
struct node {
  int data;
  struct node *next;
};
  struct node *head = NULL;
void display () {
   struct node *ptr = head;
  if ( ptr == NULL) {
   printf ( "list is empty");
    return;
  }
  printf (" Elements are");
  while ( ptr != NULL) {
   printf ("%d", ptr -> data);
   ptr = ptr -> next;
  }
   printf ("\n");
void insert_begin() {
   struct node * temp;
  temp = (struct node *) malloc (sizeof (struc
  node));
  printf (" Enter value to be inserted");
  scanf ("%d", &temp -> data);
  temp -> next = head;
  head = temp;

void insert_end() {
  struct node * temp, *ptr;
```

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

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

int main() {
 int choice;
while(1) {
printf(" 1. Insert at beginning, 2. Insert
 at end 3. Insert at any position,
 4. Display, 5. Exit");
printf(" Enter your choice");
scanf("%d", &choice);
switch(choice) {
case 1:
    insert_begin();
    break;
case 2:
    insert_end();
    break;
case 3:
    insert_pos();
    break;
Case 4:
    display();
    break;
Case 5:
    exit(0);
    break;
```

```
default:
    printf("Enter correct choice");
    }
}
return 0;
}
```

Output:

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

Enter your choice: 1
Enter the value to be inserted: 9

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

Enter your choice: 1
Enter value: 5

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

Enter your choice: 2
Enter value: 6

Date: / /

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

Enter choice: 4
Elements are 5 9 6

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

Enter your choice: 3
Enter position: 2
Enter value: 8

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

Enter choice: 4
Elements are 5 9 8 6

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

Enter your choice: 5

1/11/24

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

```
#include <stdlib.h>
typedef struct {
int * stack;
int * minstack;
int top
} Minstack;

Minstack* minstackCreate() {
Minstack* stack = (Minstack*) malloc (sizeof(
Minstack));
stack -> stack = (int *) malloc (sizeof(int)*
10000);
stack -> minStack = (int *) malloc ((sizeof(int)*
10000);
stack -> top = -1;
return stack;
}
void minstackPush (Minstack* obj, int val){
obj -> top ++;
obj -> stack [obj -> top] = val;
if (obj -> top == 0 || val <= 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];
}
void minStackFree(MinStack* obj)
{
free(obj->stack);
free(obj->minStack);
free(obj);
}
```

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

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

Output:

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

Expected:

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

18/1/24 LAB PROGRAM: 5 SINGLY LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
  int data;
  struct node * next;
};
struct node * head = NULL;
void display () {
printf(" Elements ");
struct node * ptr = head;
while (ptr != NULL)
{
printf("%d ->", ptr -> data);
 ptr = ptr -> next;
}
 printf("NULL \n");
}

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

void delete begin()
{
 if( head == NULL)
{
```

Date: / /

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

```
}
void delete_at_position()
{
 int position;
 printf(" Enter the position to delete:");
 scanf("%d", &position);
if(head == NULL)
 {
printf("List is empty");
 return;
 }
 struct node * temp, *prev;
 temp = head;
if( position == 0)
 {
 head = head -> next;
printf(" Element at position %d deleted", position);
 free(temp);
 return;
 }
for( int i=0; temp != NULL && i < position; i++)
 {
 prev = temp;
 temp = temp -> next;
}
if(temp == NULL)
{
printf(" Position is out of bounds", position);
return;
}
prev -> next = temp -> next;
printf(" Element deleted", position);
```

```
free(temp);
int main()
{
while(1)
{
printf("1. Insert at beginning\n 2. delete
beginning, 3. delete at end, 4.
delete at any position, 5. to display
6. exit\n");
printf(" Enter your choice:");
scanf("%d", &choice);
switch(choice)
{
case 1:
insert_begin();
break;
Case 2:
delete_begin();
break;
case 3:
delete_end();
break;
Case 4: delete_at_position();
break;
case 5: displaying();
break;
case 6: exit(0);
break;
default: printf(" Enter correct choice\n");
break;
}
return 0;
}
```

Output:

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

Enter your choice: 1
Enter value to insert: 11

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

Enter your choice: 1
Enter value to insert: 12

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

Enter choice: 1
Enter value to insert: 13

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

Date: / /

Enter value: 14

1.
2.
3.
4.
5.
6.

Enter choice: 1

enter value to insert: 15

1.
2.
3.
4.
5.
6.

Enter choice: 2

Element deleted from beg: 15

1.
2.
3.
4.
5.
6.

Enter choice: 3

Element deleted from end: 11

1.
2.
3.
4.
5.
6.

Enter choice: 4

Enter position to delete: 2

Element at position 2 deleted

1.
2.
3.
4.
5.
6.

Enter choice: 5

Elements are: 4 → 3 → NULL

1.
2.
3.
4.
5.
6.

Enter choice: 6

papergrid

Date: / /

18/1/24 LEET CODE: Singly linked list

```
struct listNode* reverseBetween (struct listN
start, int a, int b)
 {
  a=1;
  b=1;
struct listNode* node1=NULL, *node2=NU
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==a)
 node2=ptr;
 else if(c==b+1)
 {
 nodea=ptr;
 break;
 }
 c++;
 ptr=ptr->next;
 }
struct listNode *prc=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->next;
if (a==0)
start = ptr;
else
node(b->next = ptr;
break;
}
else
ptr = ptr->next;
}
reverse(start);
}
```

Output:

(1) head = [1,2,3,4,5]
left = 2
right = 4
output [1,4,3,2,5]
Expected [1,4,3,2,5]

(2)
Input head = [5]
left = 1
right = 1

output [5]
Expected [5]

Sneha
18/1/24

25/1/24 LAB-6 - Single linked list

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
int data;
struct node* next;
};
struct Node* createNode(int value)
{
  struct Node* newNode = (struct Node*)
malloc(sizeof(struct Node));
  newNode->data = value;
  newNode->next = NULL;
  return newNode;
}
void insertEnd(struct Node** head,
int value) {
  struct Node* newNode = createNode(value);
  if (*head == NULL) {
    *head = newNode;
  }
  else {
    struct Node* temp = *head;
    while (temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = newNode;
  }
}
void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
```

```
printf("%d->", temp->data);
    temp = temp->next;
}
printf("NULL");
}
void sortlinkedlist(struct Node* head)
{
    int swapped, i;
    struct Node *ptr1;
    struct Node *lptr = NULL;
    if(head == NULL)
        return;
do {
    swapped = 0;
    ptr1 = head;
while (ptr1->next != lptr) {
if( ptr1->data > ptr1->next->data) {
    int temp = ptr1->data;
    ptr1->data = ptr1->next->data;
    ptr1->next->data = temp;
    swapped = 1;
}
    ptr1 = ptr1->next;
}
    lptr = ptr1;
} while (swapped);
}
struct Node* reverselinkedlist(struct
                Node* head) {
struct Node* prev = NULL, *current = head,
*next = NULL;
while ( current != NULL) {
    next = current->next;
```

Date: / /

```
current->next = prev;
    prev = current;
    current = next;
}
    return prev;
}
void concatenate linked lists (struct Node* list1, struct Node* list2) {
if (*list1 == NULL) {
    *list1 = list2;
} else {
    struct Node* temp = *list1;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = list2;
}
int main() {
    struct Node *list1 = NULL;
    struct Node *list2 = NULL;
    int n, value;
    printf("Enter no. of elements 1");
    scanf("%d", &value);
    insertEnd(&list1, value);
}
    printf("Enter no for 2");
    scanf("%d", &n);
    printf("Elements 2:");
    for(int i=0; i<n; i++) {
        scanf("%d", &value);
    }
    sortedlinkedlist(list1);
```

```
printf("Sorted: ");
display(list1);
list2 = reverselinkedlist2);
printf("Reversed 2");
display(list2);
concatenatelinkedlists(&list1, list2);
printf("Concatenated list");
display(list1);
struct Node* temp;
while(list1 != NULL)
    temp = list1;
list1 = list1->next;
free(temp);
return 0;
}
```

Output:

Enter the no. of elements
Enter elements
1
2
3
4

Enter the no. of element
Enter the elements
5
6
7
8

Sorted list: 1 -> 2 -> 3 -> 4 -> NULL
Rev -> 2: 8 -> 7 -> 6 -> 5 -> NULL

Concatenated → 1 → 2 → 3 → 4 → 8 → 7 → 6 → 5 → NULL

Stack :-

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)
    malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}
void push(struct Node** top, int value) {
    struct Node* newNode = createNode(value);
    newNode->next = *top;
    *top = newNode;
}
int pop(struct Node** top) {
    if (*top == NULL) {
    printf("Underflow");
    return -1;
    }
    struct Node* temp = *top;
    int poppedValue = temp->data;
    *top = temp->next;
    free(temp);
    return poppedValue;
}
```

```
void displayStack (struct Node* top) {
  printf("Stack");
  while (top != NULL) {
    printf("%d", top->data);
    top = top->next;
  }
  printf("\n");
}
int main() {
  struct Node* top = NULL;
  int choice, value;
  do {
  printf("Stack OP");
  printf("1. Push");
  printf("2. Pop");
  printf("3. Display");
  printf("4. Exit");
  scanf("%d", &choice);
  switch(choice){
    case 1:
      printf("Enter value");
      scanf("%d", &value);
      push(&top, value);
      break;
    case 2:
      value = pop(&top);
      if (value != -1)
        printf("Popped: %d", value);
      break;
    case 3:
      displayStack(top);
      break;
```

Date: / /

```
case 4: printf("Existing");
        break;
default: printf("Invalid");
    }
} while(choice != 4);
struct Node *temp;
while(top != NULL) {
    temp = top;
    top = top->next;
    free(temp);
}
return 0;
}
```

Output

Enter choice: 1
Enter value: 3

Stack OP:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped value: 3

Stack OP:
1. Push
2. Pop

3. Display
4. Display
Enter choice: 3
stack 2|

Queue:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Queue {
    struct Node* front;
    struct Node* rear;
};
struct Node* createNode(int value) {
struct Node* newNode = (struct Node*)
malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;
return newNode;
}
struct Queue* createQueue() {
struct Queue* queue = (struct Queue*)
malloc(sizeof(struct Queue));
queue->front = queue->rear = NULL;
return queue;
}
void enqueue(struct Queue* queue,
int value) {
```

Date: / /

```
struct Node* newNode = createNode(value);
if(queue->rear == NULL){
    queue->front = queue->rear = newNode;
    return;
}
queue->rear->next = newNode;
queue->rear = newNode;
}
int dequeue(struct Queue* queue){
    if(queue->front == NULL){
    printf("Underflow\n");
    return -1;
}
struct Node* temp = queue->front;
int dequeued_value = temp->data;
queue->front = temp->next;
if(queue->front == NULL){
    queue->rear = NULL;
}
free(temp);
return dequeued_value;
}
void display(struct Queue* queue){
    struct Node* temp = queue->front;
    printf("Que:");
    while(temp != NULL){
        printf("%d", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main(){
    struct Queue* queue = createQueue();
```

```
int choice, value;
do {
    printf(" Queue OP ");
    printf(" 1. Enqueue ");
    printf(" 2. Dequeue ");
    printf(" 3. Display ");
    printf(" 4. Exit ");
    printf(" Enter choice ");
    scanf(" %d", &choice);
    switch (choice) {
      case 1:
        printf(" Enter value to enque ");
        scanf(" %d", &value);
        enqueue (queue, value);
        break;
      case 2:
        value = dequeue (queue);
        if (value != -1)
        printf(" Deq value %d \n", value);
      }
        break;
      case 3:
        displayQueue (queue);
        break;
      case 4:
        printf(" Exiting ");
        break;
      default:
        printf(" Invalid ");
    }
    while (choice != 4);
    Struct node * temp;
    while (queue -> front != NULL) {
```

```
temp = queue->font;
queue->font = queue->font->next;
free(temp);
}
free(queue);
return 0;
}
```

Output:

Enter choice: 1
Enter value to enqueue: 3

Queue op:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 2
Dequeue value: 1

Queue op
1. Enqueue
2. Dequeue
3. Display
4. Exit
choice: 3
Queue: 2 3

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

choice : 4
Exiting

25/1/24

papergrid

Date: / /

8/2/24 LAB-7

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *prev;
    struct node *next;
};
    struct node *sl = NULL;
    struct node *createNode (int value)
{
    struct node **temp = (struct node*)
     malloc (sizeof (struct node));
  temp -> data = value;
   temp -> next = NULL;
   temp -> prev = NULL;
    return temp;
}
    struct node *insert left (struct node*
                              start)
{
    int value, key;
    struct node *temp = createNode (0);
    printf(" Enter value to insert");
    scanf("%d", &temp -> data);
    printf(" Enter value to insert at end");
    scanf("%d", &key);
    Struct node *ptr = start;
    while (ptr != NULL && ptr -> data != key)
{
    ptr = ptr -> next;
}
```

```
if (ptr == NULL)
{
printf(" Node not found \n", inp);
free (temp);
}
else
{
temp -> next = ptr;
temp -> prev = ptr -> prev;
if( ptr -> prev != NULL)
{
ptr -> prev -> next = temp;
}
ptr -> prev = temp;
if (ptr == start)
{
start = temp;
}
}
return start;
}
struct node * delete value ( struct node * start)
{
int value;
printf(" Enter value to delete");
scanf(" %d", &value);
struct node * ptr = start;
while (ptr != NULL && ptr -> data != value)
{
ptr = ptr -> next;
}
if (ptr == NULL)
```

```
{
printf(" Node not found", value);
}
else
{
if( ptr -> prev != NULL)
{
ptr -> prev -> next = ptr -> next;
}
else
{
start = ptr -> next;
}
if ( ptr -> next != NULL)
{
ptr -> next -> prev = ptr -> prev;
}
printf(" Node deleted");
free (ptr);
}
return start;
}
void display (struct node * start)
{
struct node * ptr = start;
if (start == NULL)
{
printf(" Empty");
}
else
{
printf(" list ");
while (ptr != NULL)
```

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

```
default:
    printf("Invalid");
    }
}
return 0;
}
```

output:

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

1

Doubly linked list created

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

} Repeat same for the rest.

2

Enter value to insert: 11
Enter value to the left: 0

1.
2.
3.
4.
5.

2

Enter value to insert: 12

Enter value to left: 11

2

Enter value to insert: 13
Enter value to left: 12

3

Enter value to delete: 12
Node with value 12 deleted.

4

List contents:
13
11
10

5

Leetcode. - Split linked list in parts.

```
#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 ListNod
    = malloc(k* size of (struct ListNod

    current = head;
    for (int i=0; i<k; i++)
    {
        struct ListNode* part_head = current;
    int part_length = part_size + (i < extra_n
                        : 0);
for(int j=0; j < part_length - 1 && current;j
    {
        current = current->next;
    }
    if( current)
    {
    struct ListNode* nextnode = current->
```

```
current->next = NULL;
result[i] = past_head;
current = next_node;
}
else
{
result[i] = NULL;
}
}
```

output:

head = [1,2,3]

i <= 5

output:

[[1],[2],[3],[],[]]

Expected

[[1],[2],[3],[],[]]

1/8/24

15/2/24 LAB: 8 : Binary Search Tree.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode *right;
};
struct TreeNode* createNode(int data){
struct TreeNode* newNode = (struct
(TreeNode*) malloc (sizeof (struct
                        TreeNode));
NewNode->data = data;
newNode->left = newNode->right = NULL;
    return Node;
}
struct TreeNode* insertNode (struct
        TreeNode* root, int data) {
if (root == NULL) {
    return createNode (data);
}
if (data < root->data) {
    root->left = insertNode (root->left,
                        data);
}
else if (data > root->data) {
    root->right = insertNode (root->right,
                        data);
} return root;
}
void inOrderTraversal (struct
        TreeNode* root) {
```

```
if(root != NULL) {
inorderTraversal(root->left);
printf("%d", root->data);
inorderTraversal(root->right);
}}
void preorderTraversal(struct TreeNode*
                       root) {
if(root != NULL) {
printf("%d", root->data);
preorderTraversal(root->left);
preorderTraversal(root->right);
}}
void postOrderTraversal(struct TreeNode*
                        root) {
if(root != NULL) {
postorderTraversal(root->left);
postorderTraversal(root->right);
printf("%d", root->data);
}}
void displayTree(struct TreeNode* root) {
printf("\n inorder:");
inorder(root);
printf("\n");
printf("\n Pre-order");
preorder(root);
printf("\n");
printf("\n Post-order");
postorder(root);
printf("\n");
}

int main() {
struct TreeNode* root = NULL;
int choice, data;
```

```
do {
    printf("1.Insert");
    printf("2.Display");
    printf("3.Exit");
    printf("Enter your choice");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            printf("Enter");
            scanf("%d", &data);
            root = insertNode(root, data);
            break;
        case 2:
            if(root == NULL) {
                printf("Empty");
            } else {
                displayTree(root);
            }
            break;
        case 3:
            printf("Exit");
            break;
        default;
            printf("Invalid");
    }
} while (choice != 3);
return 0;
}
```

output:

1. Insert node
2. Display
3. Exit

choice: 1

Data insert: 10

1. Insert node
2. Display
3. Exit

choice: 1

Data to insert: 40

1. Insert
2. Display
3. Exit

choice: 1

Data: 55

1. Insert
2. Display
3. Exit

choice: 1

Insert: 109

1. Insert
2. Display
3. Exit

choice: 2

In-order: 10 40 55 109

Pre-order - 10 40 55 109

Post-order: 109 55 40 10

15/2/24 Leet code - linked list

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

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

k = 2

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

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

case: 2

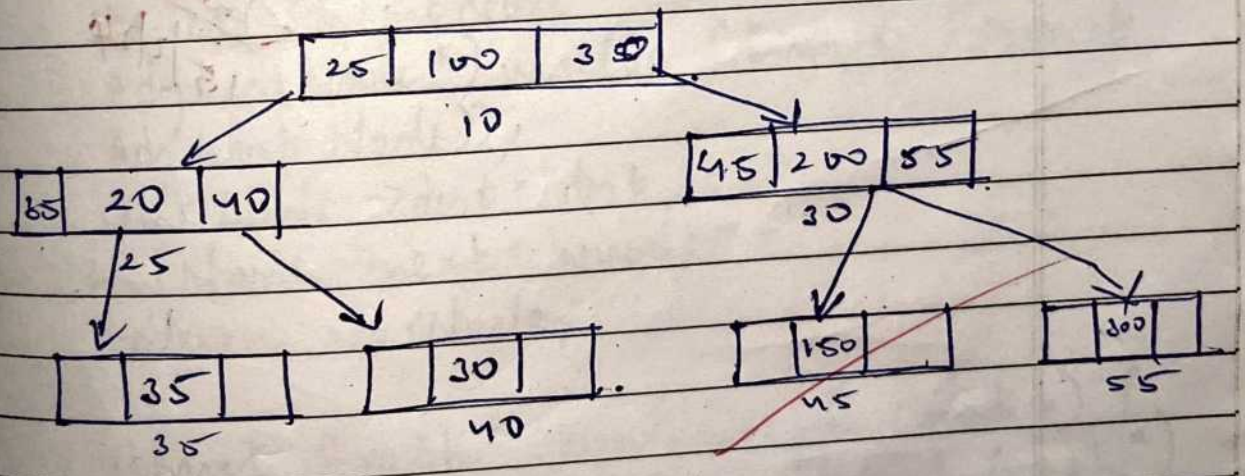
[0, 1, 2]

k = 4

output [2, 0, 1]

Expected [2, 0, 1]

TRACING of LAD —



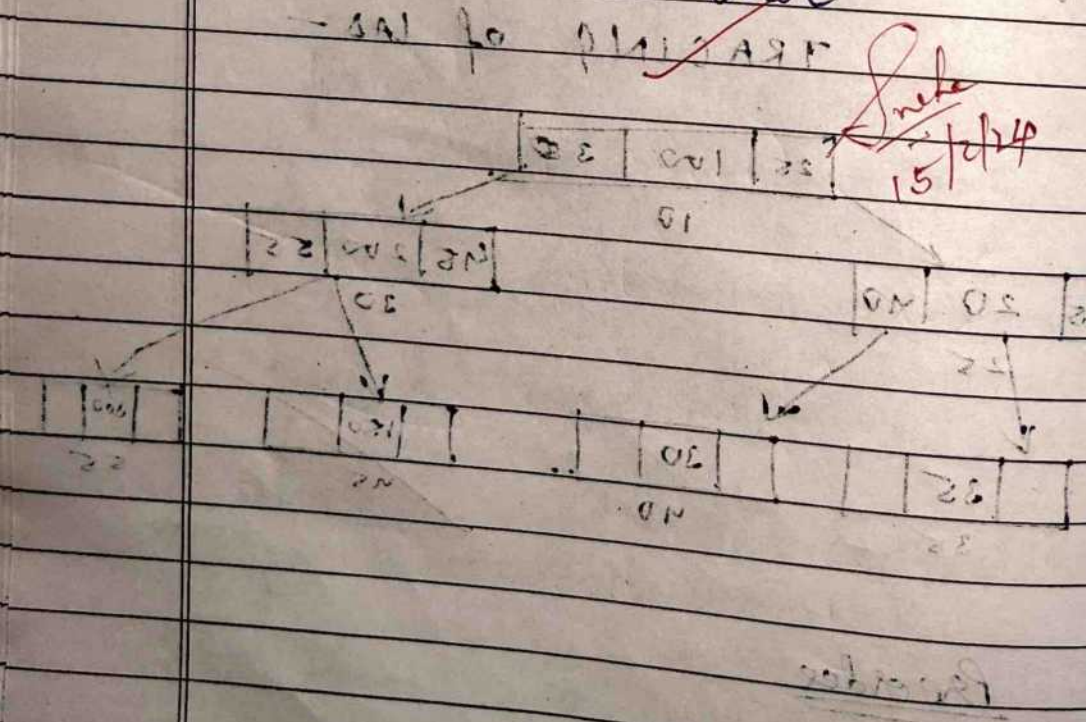
Preorder

root != NULL

Print 100

left node 25 Print 200
left node 35 Print 35
left node NULL

Right node 30 Print 250
left Right node 45 Print 150
Right node 55 Print 300
left node NULL → recursion
left node NULL right node NULL
end.



Lab: 9

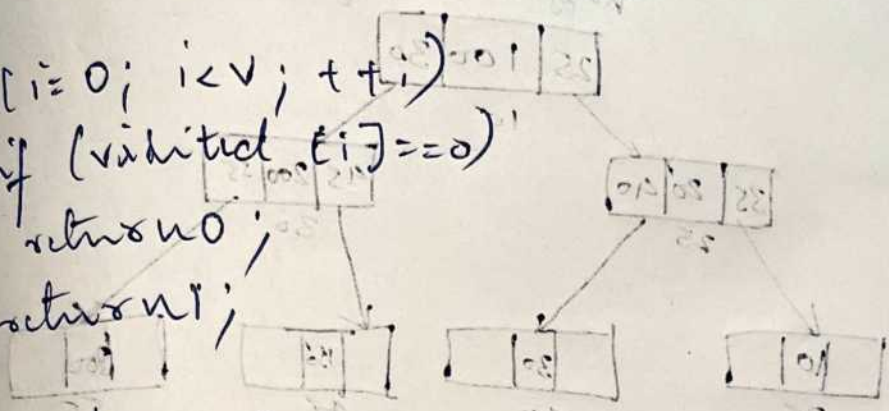
BFS & DFS:

DFS:

```
#include <stdio.h>
#include <stdlib.h>
struct AdjListNode {
    int dest;
    struct AdjListNode * next;
};
struct AdjList {
    struct AdjListNode * head;
};
struct Graph {
    int V;
    struct AdjList* array;
};

struct AdjListNode* newAdjListNode(int
                                   dest) {
    struct AdjListNode* newNode = (struct
    AdjListNode*) malloc (sizeof (struct
    AdjListNode));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph (int V) {
    struct Graph *graph = (struct Graph*)
    malloc (sizeof (struct Graph));
    graph -> V = V;
    graph -> array = (struct AdjList*) malloc
    (V * sizeof (struct AdjList));
    for (int i =0; i<V; ++i)
        graph -> array [i]. head = NULL;
```

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

void DFSUtil(struct Graph* graph,
    int v, int* visited) {
  visited[v] = 1;
  struct AdjListNode* temp = graph
                  ->array[v].head;

while(temp != NULL) {
  int adjVertex = temp->dest;
  if(!visited[adjVertex])
      DFSUtil(graph, adjVertex, visited);
  temp = temp->next;
}
}
int isConnected(struct Graph* graph,
                int v) {
int* visited = (int*) malloc(v* sizeof
                (int));
int i;
for(i=0; i<v; ++i)
```

```
visited[i]=0;
for(i=0; i<v; ++i)
if(graph->array[i].head != NULL) {
   DFSUtil(graph, i, visited);
break;
}
for(i=0; i<v; ++i)
  if (visited[i]==0)
   return 0;
   return 1;
}
  int main()
  {
  int v, E;
printf("Enter no. of vertex");
scanf("%d", &v);
struct Graph* graph= create Graph(V);
printf(" Edges");
 scanf("%d", &E);
printf(" Eges (");
 for (int i=0; i<E; ++i)
   {
     int src, dest;
scanf("%d%d", &src, &dest);
addEdge (graph, src, dest);
}
if( isConnected(graph, v))
 printf(" connected");
 else { Not connected");
```



Output:

No. of Vertices: 5
No. of Edges: 4
Enter Edges:
0 1
1 2
2 3
3 4
Connected.

01/11/24

Date __/__/__
Page _____

BFS:

```
#include <stdio.h>
#include <stdlib.h>
struct AdjListNode {
    int dest;
    struct AdjListNode* next;
};
struct AdjList {
    struct AdjListNode* head;
};
struct Graph {
    int V;
    struct AdjList* array;
};
struct AdjListNode* newAdjList
        Node(int dest) {
struct AdjListNode* newNode = (struct
AdjListNode) malloc (sizeof(struct
    AdjListNode));
    newNode->dest = dest;
    NewNode->next = NULL;
    return newNode;
}
struct Graph* createGraph(int V)
{
    struct Graph* graph = (struct
            Graph*) malloc (sizeof
(struct Graph));
    graph->V = V;
graph->array = (struct AdjList*) malloc
    (V* sizeof (struct AdjList));
```

Date __/__/__
Page

```
for (int i=0; i<v; i++)
    graph->array[i].head = NULL;
    return graph;
}
void addEdge(struct Graph* graph,
int src, int dest) {
struct AdjListNode* newNode =
    newAdjListNode(dest);
newNode->next = graph->array[src].
    graph->array[src].head = newNode;
newNode = newAdjListNode(src);
newNode->next = graph->array[dest].
    head;
graph->array[dest].head = newNode;
}
void BFS(struct Graph* graph, int
                                start){
struct int *visited = (int*) malloc
(graph->v * sizeof(int));
for (int i=0; i<graph->v; i++)
    visited[i]=0;
int* queue = (int*) malloc(graph
                    ->v * sizeof int));
int front=0, rear=0;
    visited[i]=0;
int* queue = (int*) malloc(graph
                    ->v * sizeof(int));
while (front<rear){
int currentVertex = queue[front++];
```

Date __/__/__
Page _____

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

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

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

BFS(graph, start vertex);
return 0;
}

Output

Enter vertices : 6
Enter edges : 7
Enter edges
0 1
0 2
1 2
1 3
2 3
3 4
4 5
Start vertex : 0
traversal : 0 2 1 3 4 5

Date __/__/__
Page _____

11/2/24

HACKER RANK

```
#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;
    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 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 = 1; i <=~~
```

Output:-

-1
-1
2
1
1

Expected Output

Date / /
Page

27/2/24

LAB-10: Hashing

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TABLE_SIZE 10
#define KEY_LENGTH 5
#define MAX_NAME_LENGTH 50
#define MAX_DESIGNATION_LENGTH 50
struct Employee {
    char key[KEY_LENGTH];
    char name[MAX_NAME_LENGTH];
    char designation[MAX_DESIG_LENGTH];
    float salary;
};
struct HashTable {
    struct Employee *table[Table_Size];
};
int hash_function(const char *key) {
    int sum = 0;
    for(int i=0; key[i]!='\0'; i++) {
        sum += key[i];
    }
    return sum % Table_Size;
}
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);
while (ht->table[index] != 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 filename");
scanf("%s", filename);
file = fopen(filename, "r");
if (file == NULL) {
printf("Error");
return 1;
```

```
}
while (fgets (line, sizeof (line), file)) {
emp = (struct Employee *) malloc (sizeof
                    (struct Employee));
sscanf ("%s %s %s %f", emp->key, emp->name,
        emp->desg, emp->sal);
insert (&ht, emp);
}
fclose (file);
printf (" key ");
scanf (" %s", key);
emp = search (&ht, key);
if (emp != NULL) {
printf ("
printf (" Name %s, emp->name);
printf ("Desig %s, emp->desig);
printf (" Sal %.2f\n", emp->salary);
}
else {
printf (" Not found %s, key);
}
for (int i=0; i<TABLE_SIZE; i++) {
if (ht.table[i] != NULL) {
free (ht.table[i]);
}
}
return 0;
}
```

output:

Enter file name: employee.txt

key: 1223

Employee no 1223: Name: Manya

Desg: CEO

Salary: 1,00,000.00