INDEX

Name Vivuthna. (Pdv Rajeswari) Std 3 Sec D

Roll No. — Subject DS School/College BMSCE

School/College Tel. No. 1BM22CS193 ~~Parents~~ Tel. No. 6305465096

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1	7/12/23	Lab 0		
2	21/12/23	Lab 1		
3	28/12/23	lab 2		
4	11/1/24	Lab 3		
~~5~~	~~18/1/24~~	~~Lab 4~~ ?		
5	18/1/24	lab 4		
6	25/1/24	labs, stack, Queue		
7	1/2/24	lab 6		
8	14/2/24	Lab 7, Leet code.		
9	22/2/24	Lab 8, Hacker Rank		
10	29/2/24	Lab 9		

LAB-1

Write a program to implement Dynamic Memory allocation like malloc, calloc, free and realloc

```
#include <stdio.h>
#include <stdlib.h>
void * Malloc (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 the size of the array:");
 scanf("%d", &size);
 arr1 =(int*) Malloc(size * sizeof(int));
 if (arr1 == NULL)
 {
     printf("Memory allocation failed.\n");
     return 1;
 }
```

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

Output

Enter the size of the array: 6
Enter elements of the array:
Element 1: 34
Element 2: 56
Element 3: 78
Element 4: 90
Element 5: 100
Element 6: 123
Elements of the array (malloc):
34 56 78 90 100 123
Enter additional elements of the array:
Element 7: 23
Element 8: 45
Element 9: 67
Element 10: 89
Element 11: 34
Element 12: 1234
Elements of the array (realloc):
34 56 78 90 100 123 23 45 67 89 34 1234

Date __/__/__
Page. _____

Implementation of stack using [push, pop, display function]

```
#include <stdio.h>
#include<stdlib.h>
#define MAX_SIZE 10
struct Stack {
    int items[MAX_SIZE];
    int top;
};
void initialize(struct stack* stack)
{
    stack->top = -1;
}
int inEmpty(struct Stack * stack)
{
        return stack->top == -1;
}
int isFull(struct Stack *stack()
{
        return stack -> top== MAX_SIZE -1;
}
void push(struct Stack* stack, int value)
{
    if (isFull(stack))
        {
        printf(" stack overflow. Cannot push %d\n",
                                        value);
    }
    else
    {
        Stack->top++;
        Stack->items[stack->top] = value;
        printf(" pushed %d onto the stack.\n", value);
```

```
    }
}
int pop(struct Stack * stack)
{
    int poppedvalue = -1;
    if (isEmpty(stack))
    {
        printf("stack underflow. Cannot pop from
                an empty stack.\n");
    }
    else
    {
        poppedvalue = stack->items[stack->top];
        stack->top--;
        printf("popped %d from the stack.\n",
                poppedvalue);
    }
    return poppedvalue;
}
void display(struct Stack * stack)
{
    if (isEmpty(stack))
    {
        printf("stack is empty.\n");
    }
    else
    {
        printf("elements in the stack:\n");
        for(int i=0; i<=stack->top; i++)
        {
            printf("%d\n", stack->items[i]);
        }
    }
}
```

```
int main()
{
    struct stack stack;
    initialize(&stack);
    push(&stack, 50);
    push(&stack, 100);
    push(&stack, 900);
    display(&stack);
    pop(&stack);
    display(&stack);
    push(&stack, 5);
    display(&stack);
    return 0;
}
```

O/P

Pushed 50 onto the stack
Pushed 100 onto the stack
Pushed 900 onto the stack.
Elements in the stack : 50 100 900
popped 900 from the stack
Elements in the stack : 50 100
Pushed 5 onto the stack.
Elements in the stack : 50 100 5

Sahil
2/11/22

LAB-2

Date 28/12/23
Page

Infix to Postfix

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50
char stack[SIZE];
int top = -1;
push(char ele)
{
    stack[++top] = ele;
}
char pop()
{
    return(stack[top--]);
}
int pr(char symbol)
{
    if(system == '^')
    {
      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 the infix expression:");
   scanf("%s", infix);
   push('#');
   while((ch=infix[i++])!='\0')
   {
      if(ch=='(') push(ch);
      else
       if(isalnum(ch)) postfix[k++]=ch;
       else
        if(ch==')')
        {
           while(stack[top]!='(')
              postfix[k++]=pop();
           ele=pop();
        }
        else
        {
           while(pr(stack[top])>=pr(ch))
           postfix[k++]=pop();
           push(ch);
        }
   }
   while(stack[top]!='#')
      postfix[k++]=pop();
   postfix[k]='\0';
   printf("\n Postfix expression=%s\n", postfix);
}
```

O/P

enter the infix expression: 4+(5*5-(12/3^2)*4)*3

Postfix expression: 455*1232^/4*-3*+

Date __/__/__
Page

Evaluation of postfix expression:

```
#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("In the result of expression:%s=%d\n\n",
                                    exp,pop());
}
```

O/P

enter the expression: 32*8+

the result of expression: 32*8+ = 14

Date __/__/__
Page _____

3a. Queue implementation.

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

```
{
 case 1:
 insert();
 break;
 case 2:
 delete();
 break;
 case 3:
 display();
 break;
 case 4:
 exit(0);
 break;
 default:
 printf("invalid choice\n");
 }
 }
}
insert()
{
 int add_item;
 if (rear==MAX-1)
 printf("queue overflow\n");
 else
 {
 if(front==-1)
 front=0;
 printf("insert the element in the queue\n");
 scanf("%d", &add_item);
 rear=+1;
 queue_array[rear]=add_item;
 }
}
```

```
delete()
{
if(front == -1 || front > rear)
{
printf("queue underflow\n");
return;
}
else
{
printf("deleted element is : %d\n", queue_array[
front += 1;
}
}
```

o/p

1. insert ~~the element in the queue~~ 6
2. delete
3. display
4. exit

enter your choice: 1

insert the element in the queue: 2

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

enter your choice: 1

insert the element in the queue: 4

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

enter your choice: 2

deleted element is: 2

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

enter your choice: 3

Date __/__/__
Page _____

Qveve is:
4
1. insert
2. delete
3. display
4. exit.
enter your choice: 4

3b. Circular Qveve Implementation

```
#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("\nQueue is full");
    else{
        if(front == -1)
            front = 0;
        rear = (rear+1)%SIZE;
        items[rear] = element;
        printf("\nInserted -> %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("\n Deleted element -> %d \n", element);
    return element;
    }
  }
void display(){
  int i;
  if(isEmpty())
    printf("\n Empty queue\n");
  else{
  printf("\n front -> %d", front);
  printf("\n items -> ");
  for(i = front; i != rear; i = (i+1) % SIZE){
    printf("%d", items[i]);
  }
  printf("%d", items[i]);
```

```
        printf("\nRear -> %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;
}
```

O/P

```
Inserted -> 1
Inserted -> 2
Inserted -> 3
Inserted -> 4
Inserted -> 5
Front -> 0
Items -> 1 2 3 4 5
Rear -> 4
Deleted element -> 1
Deleted element -> 2
Front -> 2
Items -> 3 4 5
Rear -> 4
```

```
Inserted -> 6
Inserted -> 7
Front -> 2
Items -> 3 4 5 6 7
Rear -> 4
```

11/1/24

LAB-3

Date __/__/__
Page _____

```
#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\n");
        return;
    }
    printf("Elements are: ");
    while (ptr != NULL){
        printf("List is empty\n");
        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(struct node));
    printf("Enter the value to be inserted: ");
    scanf("%d", &temp->data);
    temp->next = head;
    head = temp;
}
```

Date __/__/__
Page _____

```
void insert_end() {
    struct node *temp, *ptr;
    temp = (struct node *) malloc (sizeof(struct node));
    printf("Enter the value to be inserted: ");
    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 the position to insert: ");
    scanf("%d", &pos);
    printf("Enter the value to be inserted: ");
    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\n");
            return;
        }
    }
    temp->next = ptr->next;
    ptr->next = temp;
  }
}
int main() {
    int choice;
    while(1) {
        printf("\n1. Insert at the beginning
                \n2. Insert at the end
                \n3. Insert at any position
                \n4. Display
                \n5. Exit\n");
        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 the correct choice\n");
    }
}
return 0;
}
```

O/P

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

Enter your choice: 1
Enter the value to be inserted: 2.
1. Insert at the beginning
2. Insert at the end
3. Insert at any position.
4. Display
5. Exit

Enter your choice: 1
Enter the value to be inserted: 4
1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit

Enter your choice: 2
Enter the value to be inserted: 6

Date __/__/__
Page _____

1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice: 4
Elements are: 4 26
1. Insert at the beginning
2. Insert at the ~~beginning~~ end
3. Insert at any position
4. ~~Insert~~ Display
5. ~~Insert~~ Exit
Enter your choice: 3
Enter the position to insert: 2
Enter the value to be inserted: 7
1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice: 4
Elements are: 4 2 7 6
1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice: 5

Sate
11/12/24

Leet code - Min stack

Date __/__/__
Page _____

```
#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];
}

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

~~Op~~ ~~Input~~

["MinStack", "push", "push", "push", "getMin", "pop", "top"

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

~~output~~

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

Expected

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

16/1/24

LAB-4

Date __/__/__
Page _____

```
#include<stdio.h>
#include<stdlib.h>
Struct node {
    int data;
    struct node *next;
};
Struct node *head = NULL;
void display() {
    printf("Elements are: ");
    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 the value to be inserted: ");
    scanf("%d", &temp->data);
    temp->next = head;
    head = temp;
}
void delete_begin() {
    if (head == NULL) {
        printf("List is empty. Deletion not possible.\n");
        return;
    }
    struct node *temp = head;
    head = head->next;
    printf("Element deleted from the beggining: %d\n",
        temp->data);
    free(temp);
}
```

"getMin"

```
void delete_end(){
    if(head == NULL){
        printf("List is empty. Deletion is not possible");
        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 from the end: %d\n",
            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. Deletion not possible");
        return;
    }
    struct node *temp, *prev;
    temp = head;
    if(position == 0){
        head = head->next;
        printf("Element at position %d deleted
                successfully.\n", position);
```

```
        free(temp);
        return;
    }
    for (int i = 0; temp != NULL && i < position; i++) {
        prev = temp;
        temp = temp -> next;
    }
    if (temp == NULL) {
        printf("Position %d is out of bounds.\n", position);
        return;
    }
    prev -> next = temp -> next;
    printf("Element at position %d deleted successfully.
    .\n", position);
    free(temp);
}
int main() {
    int choice;
    while (1) {
        printf("\n1. to insert at the beginning\n
            2. to delete beginning\n 3. to
            delete at end\n 4. to delete at
            any position\n 5. to display\n 6. to 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:
            display();
            break;
        case 6:
            exit(0);
            break;
        default:
            printf("Enter the correct choice\n");
            break;
        }
    }
    return 0;
}
```

O/P:

```
1. to insert at the beginning
2. to delete beginning
3. to delete end
4. to delete at any pos
5. to display
6. to exit
Enter your choice: 2
Enter the value to be inserted: 2
1. to insert at the beginning
2. to delete at beginning
3. to delete at end
4. to delete at any pos
5. to display
6. to exit
```

repeat option 2 for values 3,4,5,6

Date __/__/__
Page _____

Enter your choice: 2
Element deleted from the beginning: 6
1. to insert at the beginning
2. to delete beginning
3. to delete end
4. to delete at any pos
5. to display
6. to exit
Enter your choice: 3
Element deleted from the end: 2
1. to insert at the beginning
2. to delete beginning
3. to delete end
4. to delete at any pos
3. to display
6. to exit
Enter your choice: 4
Enter the position to delet: 2
Element at position 2 deleted successfully
1. to insert at the beginning
2. to delete beginning
3. to delete end
4. to delete at any pos
5. to display
6. to exit
Enter your choice: 5
Elements are: 5 -> 4 -> NULL
1. to insert at the beginning
2. to delete beginning
3. to delete end
4. to delete at any pos
5. to display
6. to exit
Enter your choice: 6

Leet code - reverse linked list II

Date / /
Page

```
struct ListNode* reverseBetween(struct ListNode* head,
int a, int b)
{
a--;
b--;
struct ListNode
*node1 = NULL, *node2 = NULL, *nodeb = NULL, *nodea =
*ptr = head;
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 *pre = nodea, *temp;
ptr = head;
c = 0;
while(ptr != NULL)
{
if(c >= a && c < b)
{
```

```
temp = ptr -> next;
ptr -> next = pre;
pre = ptr;
ptr = temp;
}
else if (c == k)
{
ptr -> next = pre;
if (a == 0)
start = ptr;
else
nodeb -> next = ptr;
break;
}
else
ptr = ptr -> next;
c = c + 1;
}
return start;
}
```

NULL,

Output

case 1	case 2
Input head = [1,2,3,4,5] left = 2 right = 4 output [1,4,3,2,5] Expected [1,4,3,2,5]	Input head = [5] left = 1 right = 1 Output [5] Expected [5].

25/1/24 LAB-5

Date __/__/__
Page _____

```
#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\n");
}
```

```
void SortLinkedList(Struct Node* head){
    int Swapped, i;
    Struct Node* ptr;
    Struct Node* lptr=NULL;
    if(head==NULL)
        return;
    do {
        Swapped=0;
        ptr=head;
        while(ptr->next != lptr){
            if(ptr->data > ptr->next->data){
                int temp=ptr->data;
                ptr->data=ptr->next->data;
                ptr->next->data=temp;
                Swapped=1;
            }
            ptr=ptr->next;
        }
        lptr=ptr;
    } while(Swapped);
}
Struct Node* reverseLinkedList(struct Node* head){
    struct Node *prev=NULL, *current=head, *next=NULL;
    while(current!=NULL){
        next=current->next;
        current->next=prev;
        prev=current;
        current=next;
    }
    return prev;
}
```

```
void concatenateLinkedLists(struct Node** list1, struct Node
    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 the no. of elements for list1:");
    scanf("%d", &n);
    printf("Enter the elements for list1:\n");
    for(int i=0; i<n; i++){
        scanf("%d", &value);
        insertEnd(&list1, value);
    }
    printf("Enter the number of elements for list2:");
    scanf("%d", &n);
    printf("Enter the elements for list2:\n");
    for(int i=0; i<n; i++){
        scanf("%d", &value);
        insertEnd(&list2, value);
    }
    SortedLinkedList(list1);
    printf("Sorted list1:");
    display(list1);
    list2 = reverseLinkedList(list2);
    printf("Reverse list2:");
```

```
list 2){   display(list 2);
           Concatenate LinkedLists(&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 number of elements for ~~list~~ list 1: 4
Enter the elements for list 1:
7
5
9
3
Enter the number of elements for list 2: 4
Enter the elements for list 2:
9
3
4
6
Sorted list 1: 3 → 5 → 7 → 9 → NULL
Reversed list 2: 6 → 4 → 3 → 9 → NULL
Concatenate list: 3 → 5 → 7 → 9 → 6 → 4 → 3 → 9 → NULL

Stack

Date __/__/__
Page _____

```
#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 No
    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("Stack underflow!\n");
        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("\n Stack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to push: ");
                scanf("%d", &value);
                push(&top, value);
                break;
            case 2:
                value = pop(&top);
                if (value != -1) {
                    printf("popped value: %d\n", value);
                }
                break;
            case 3:
                displayStack(top);
                break;
            case 4:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice!\n");
```

```
    }
} while (choice != 4);
struct Node* temp;
while (top != NULL) {
    temp = top;
    top = top->next;
    free(temp);
}
return 0;
}
```

Stack operations:
1. push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 2
stack operations:
1. push
2. pop
3. Display
4. exit
Enter your choice: 1
Enter the value to push: 3
Stack operations:
1. push
2. pop
3. Display
4. exit
Enter your choice: 1
Enter the value to push: 4
Stack operations

1. Push
2. pop
3. Display
4. Exit
Enter your choice: 2
Popped value: 4

Stack operations:
1. Push
2. pop
3. Display
4. Exit
Enter your choice: 3
Stack: 3 2

Stack operations:
1. Push
2. pop
3. Display
4. exit
Enter your choice: 4
Exiting the pgm

Queue

Date __/__/__
Page _____

```
#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){
    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("Queue underflow!\n");
        return -1;
    }
    struct Node* temp = queue->front;
    int dequeuedValue = temp->data;
    queue->front = temp->next;
    if (queue->front == NULL){
        queue->rear = NULL;
    }
    free(temp);
    return dequeuedValue;
}
void displayQueue(struct Queue* queue){
    struct Node* temp = queue->front;
    printf("Queue: ");
    while (temp != NULL){
        printf("%d", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main(){
    struct Queue* queue = createQueue();
    int choice, value;
    do{
        printf("\nQueue Operations:\n");
        printf("1. Enqueue\n");
        printf("2. dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice){
    case 1:
        printf("Enter the value to enqueue:");
        scanf("%d", &value);
        enqueue(queue, value);
        break;
    case 2:
        value = dequeue(queue);
        if (value != -1){
            printf("Dequeued value: %d\n", value);
        }
        break;
    case 3:
        display Queue(queue);
        break;
    case 4:
        printf("Exiting the program\n");
        break;
    default:
        printf("Invalid Choice\n");
    }
} while (choice != 4);
struct Node* temp;
while (queue->front != NULL){
    temp = queue->front;
    queue->front = queue->front->next;
    free(temp);
}
free(queue);
return 0;
}
```

Date / /
Page

Op Queue operations:
1. Enqueue
2. dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 2

repeat for values 3, 4 & 5 by choosing option 1

Queue operations
1. Enqueue
2. dequeue
3. Display
4. Exit
Enter your choice 2.
Dequeued value: 2

Queue operations
1. Enqueue
2. dequeue
3. Display
4. Exit
Enter your choice: 3
Queue: 3 4

Queue operations
1 Enqueue.
2 dequeue
3 Display
4 Exit
Enter your Choice: 4
Exiting the program.

11/12/24 Lab - 6 Doubly linked list
Delete, Insert, Traversal Date / /
Page

```
#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 *insertleft(struct node *start){
    int value, key;
    struct node *temp=createNode(0);
    printf("Enter the value to be inserted:");
    scanf("%d", &temp->data);
    printf("Enter the value to the left of which the
        node has to be inserted:");
    scanf("%d", &key);
    struct node *ptr = start;
    while(ptr!=NULL && ptr->data!=key){
        ptr=ptr->next;
    }
    if(ptr==NULL){
    printf("Node with value %d not found\n", key);
    free(temp);
    } else{
    temp->next = ptr;
```

Date __/__/__
Page _____

```
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 the value to delete:");
scanf("%d", &value);
struct node *ptr = start;
while(ptr!=NULL && ptr->data!=value)
    ptr = ptr->next;
}
if (ptr==NULL){
    printf("Node with value %d not found\n", 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 with value %d deleted\n", value);
free(ptr);
}
```

```
    return start;
}
void display(struct node *start){
    struct node *ptr = start;
    if(start==NULL){
        printf("List is empty\n");
    }else{
        printf("List contents:\n");
        while(ptr!=NULL){
            printf("%d\n", ptr->data);
            ptr=ptr->next;
        }
    }
}
int main(){
    int choice;
    while(1){
    printf("\n 1. Creat a doubly linked list\n
            2. Insert to the left of a node\n
            3. Delete based on a value\n
            4. Display the contents\n 5. Exit\n");
    Scanf("%d", &choice);
    Switch(choice){
    case 1:
        s1 = createNode(0);
        printf("Doubly linked list created\n");
        break;
    Case 2:
        s1 = insert left(s1);
        break;
    Case 3:
        s1 = delete value(s1);
        break;
```

```
Case 4:
    display(ch);
    break;
Case 5:
    printf("Exited the program");
    exit(0);
default:
    printf("Invalid choice\n");
}
}
return 0;
}
```

O/P

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

1

Doubly linked list created

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

2

repeat for (77,66), (88,77), (99,88).

Enter the value to be inserted: 66

Enter the value to the left of which the node has to be inserted: 0

Date __/__/__
Page _____

1. Create
2. Insert
3. Delete
4. Display
5. exit
4
List contents:
99
88
77
66
0
1. Create
2. Insert
3. Delete
4. Display
5. exit
3
Enter the value to be deleted: 88
Node with value 88 deleted.
1. Create
2. Insert
3. Delete
4. Display
5. Exit.
4
List contents
99
77
66
0

Date __/__/__
Page _____

Leet code,

```
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 List** result = (struct ListNode**)
        malloc(k * sizeof(struct
            ListNode*));
    current = head;
    for (int i = 0; i < k; i++){
        Struct ListNode* part_head = current;
        int part_length = part_size + (i < extra_
            nodes ? 1 : 0);
        for (int j = 0; j < part_length - 1 &&
            current; j++){
            current = current->next;
        }
        if (current){
            Struct List* next_node = current->next;
            current->next = NULL;
            result[i] = part_head;
            current = next_node;
        }else{
            result[i] = NULL;
        }
    } *returnSize = k;
    return result;
}
```

Date __/__/__
Page _____

Case 1

input

[1, 2, 3]

k =

5

output

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

Case 2

input

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

k =

3

output

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

8/4/2

1/2/24

14/2/24

LAB - 7

BINARY SEARCH TREE

Date __/__/__
Page ____

```
#include <stdio.h>
#include <stdlib.h>
struct Tree_Node{
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* createNode(int data){
    struct TreeNode* newNode = (struct TreeNode*)
                                malloc(sizeof(struct Tree
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
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("In-order traversal: ");
    inOrderTraversal(root);
    printf("\n");
    printf("Pre-order traversal: ");
    preOrderTraversal(root);
    printf("\n");
    printf("Post-order traversal: ");
    postOrderTraversal(root);
    printf("\n");
}
int main() {
    struct TreeNode* root = NULL;
    int choice, data;
    do {
        printf("1. Insert a node\n");
        printf("2. Display tree\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch(choice) {
    case 1:
        printf("Enter data to insert: ");
        scanf("%d", &data);
        root = insertNode(root, data);
        break;
    case 2:
        if (root == NULL) {
            printf("Tree is empty.\n");
        } else {
            displayTree(root);
        }
        break;
    case 3:
        printf("Exiting pgm\n");
        break;
    default:
        printf("Enter a valid choice\n");
}
} while (choice != 3);
return 0;
}
```

request to insert 10, 30, 20, 100, 150, 200, 300

1. Insert a node	1. Insert a node
2. Display tree	2. Display tree
3. exit	3. exit
Enter your choice: 1	Enter your choice: 3
Enter data to insert: 100	Exiting pgm
1. insert a node	
2. Display tree	
3. exit	
Enter your choice: 2	

In-order traversal: 10 20 30 100 150 200 300

Pre-order traversal: 100 20 10 30 200 150 300

Post order traversal: 10 30 20 150 300 200 100

Date __/__/__
Page _____

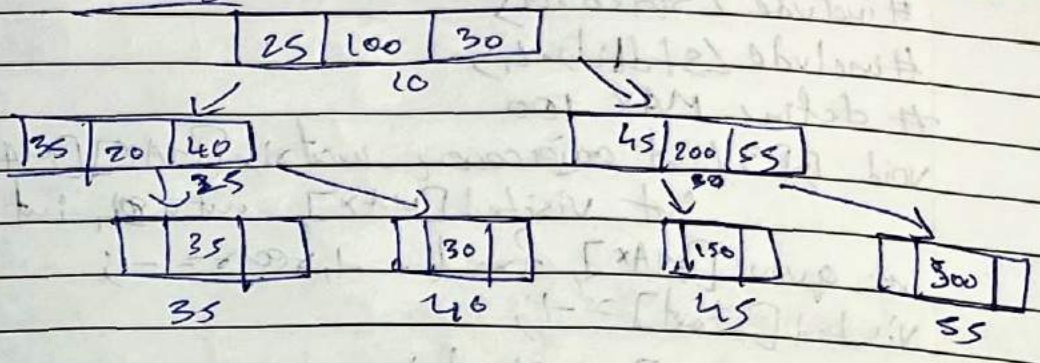
LEETCODE

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

qp)

Case 2	Case 1
head	head
[0,1,2]	[1,2,3,4,5]
k = 4	k = 2
output	output
[2,0,1]	[4,5,1,2,3]
expected	Expected
[2,0,1]	[4,5,1,2,3]

Tracing :



root = NULL

print 100 | print 20

left node 25 | print 35

left node 35 | → recursive

left node NULL

Right node 30 | print 200

left node 45 | print 150

Right node 55 | print 300

left node NULL | → recursive

left node NULL right node NULL

end

21/12/24

22/2/24 BFS

Date __/__/__
Page _____

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
void BFS(int adjacency_matrix[MAX][MAX],
         int visited[MAX], int n, int start){
int queue[MAX], front = -1, rear = -1;
visited[start] = 1;
queue[++rear] = start;
while(front != rear){
 int current = queue[++front];
printf("%d ", current + 1);
for(int i = 0; i < n; i++){
if (adjacency_matrix[current][i] == 1 && !visited
visited[i] = 1;
queue[++rear] = i;
}
}
}
}

int main(){
int adjacency_matrix[MAX][MAX], visited[MAX], n, i, j;
printf("Enter the number of nodes: ");
scanf("%d", &n);
printf("Enter the adjacency matrix:\n");
for(i=0; i<n; i++){
visited[i]=0;
for(j=0; j<n; j++)
scanf("%d", &adjacency_matrix[i][j]);
}
}

int start_node;
Printf("Enter the starting node for BFS traversal (1 to
scanf("%d", &start_node);
Printf("BFS Traversal starting from node %d: ", start_node
return 0; BFS(adjacency_matrix, visited, n, start_node-1);
```

Output:
Enter the number of nodes: 4
Enter the adjacency matrix
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Enter the starting node for BFS traversal (1 to 4): 3
BFS Traversal starting from node 3: 3 1 4 2

DFS

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000
void DFS(int adjacency_matrix[MAX][MAX], int visited[MAX], int n,
int current);
{
visited[current] = 1;
for(int i = 0; i < n; i++){
if(adjacency_matrix[current][i] == 1 && !visited[i]){
DFS(adjacency_matrix, visited, n, i);
}
}
}
int isConnected(int adjacency_matrix[MAX][MAX], int visited
[MAX], int n){
for(int i=0; i<n; i++){
visited[i]=0;
}
DFS(adjacency_matrix, visited, n, 0);
for(int i=0; i<n; i++){
if(!visited[i]){
return 0;
}
}
return 1;
}
```

NP 22/2/24

Date __/__/__
Page _____

```
int main(){
int adjacency_matrix[MAX][MAX], visited[MAX], n, i, j;
printf("Enter the number of nodes: ");
scanf("%d", &n);
printf("Enter the adjacency matrix:\n");
for(i=0; i<n; i++){
for(j=0; j<n; j++){
scanf("%d", &adjacency_matrix[i][j]);
}
}
if(isConnected(adjacency_matrix, visited, n)){
printf("The graph is connected.\n");
}else{
printf("The graph is not connected.\n");
}
return 0;
}
```

O/P

```
Enter the number of nodes:4
Enter the adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
The graph is Connected.
```

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);
    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);
}
```

Q) D)

Input	Expected
3 2 3 1 -1 -1 -1 2 1 1	3 1 2 3 1 3

22/2/24

29/2/24

Lab 9: Hashing

Date __/__/__
Page _____

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TABLE_SIZE 100
#define KEY_LENGTH 5
#define MAX_NAME_LENGTH 56
#define MAX_DESIGNATION_LENGTH 50
struct Employee {
    char key[KEY_LENGTH];
    char name[MAX_NAME_LENGTH];
    char designation[MAX_DESIGNATION_LENGTH];
    float salary;
};
struct HashTable {
    struct Employee* table[TABLE_SIZE];
};
int hash_function(const char* key, int m) {
    int sum = 0;
    for (int i = 0; key[i] != '\0'; i++) {
        sum += key[i];
    }
    return sum % m;
}
void insert(struct HashTable* ht, struct Employee* emp) {
    int index = hash_function(emp->key, TABLE_SIZE);
    while (ht->table[index] != NULL) {
        index = (index + 1) % TABLE_SIZE;
    }
    ht->table[index] = emp;
}
struct Employee* search(struct HashTable* ht, const ch
    int index = hash_function(key, TABLE_SIZE);
    while (ht->table[index] != NULL) {
        if (strcmp(ht->table[index]->key, key) == 0)
```

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

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

O/P

```
Enter the filename containing employee records: employee.txt
Enter the key to search: 1234
Employee record found with key '1234':
Name: Vinutha
Designation: reporter
Salary: 100000.00
```

Sahana
29/2/24