INDEX

NAME: Nithin R Patil STD.: _____ SEC.: 3D ROLL NO.: _____ SUB.: DS observations

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
	7/12/23	week 1 introduction	10	
	21/12/23	① Swapping numbers		
		② dynamic memory allocation	10	
		③ stack operations		
	28/12/23	① infix to postfix	10	
		② postfix evaluation		
	11/1/24	① circular queue	10	
		② Singly linked list		
	18/1/24	① deletion of Singly linked list	10	
		② leetcode 155 minstack	5	
	25/1/24	① operations on Singly list	10	
		② linked list as stack		
		③ linked list as queue.		
	1/2/24	① doubly linked list	10	
	5/2/24	② Binary Search Tree	10	
	22/2/24	① BFS		
		② DFS + Hackerrank		
	29/2/24	① Hashing		

7/12/23

1) enter username: [redacted] Nithin R Patil
enter Pwd: 1234
enter amt to deposit: 9000
Acc created successfully.

enter amount to withdraw: 7000
7000 withdrawed Successfully!
remaining balance: 2000

2) enter strings: Nithin / Abhinav / Rishi
Sorted order: Abhinav Nithin Rishi

3) enter Array: 123 456 789
enter Key: 6
Key found at pos: 1, 2

4) enter string: "Nithin"
enter substring: "hin"
Matching found at 3.

5) enter array: 4 5 6 2 3 4
enter Key: 3
Last occurance found at 4.

6) enter array: 4 3 1 8 2
enter Key: 1
Key found at 2

7) enter array: 1 2 3 4 5 6
enter Key: 3
Key found at 2

(8) enter array: 4 3 5 1 2 6
Max elemt is 6; Min element is 1.

week 2

(1) Swapping using pointers

```
#include <stdio.h>
void swap (int *x, int *y) {
 int t;
 int t = *x;
     *x = *y;
     *y = t;
}

int main () {
int *x, *y;
printf ("enter two numbers that has to be swapped:");
scanf ("%d %d", &x, &y);
printf ("x = %d and y = %d", x, y);
swap (&x, &y);
printf ("\n after swapping x = %d and y = %d", x, y);
return 0;
}
```

output

```
enter two number that has to be swapped: 1
2
x = 1 and y = 2
after swapping x = 2 and y = 1
```

(2) Malloc()

```
ptr = (int *) malloc(n * sizeof(int));
```

calloc()

```
ptr = (int *) calloc(n, sizeof(int));
```

free()

```
free(ptr);
```

realloc()

~~ptr = (int *) malloc(~~

~~ptr = (int *)~~

~~ptr = 2~~

```
ptr = (int *) realloc(ptr, n * sizeof(int));
```

```
#include <stdio.h>
int main() {
int * ptr;
int i, n, m;
printf("enter no of elements:");
scanf("%d", &n);
ptr = (int *) malloc(n * sizeof(int));
if (ptr == NULL) {
    printf("memory not allocated by malloc");
}
else {
    for(i=0; i<n; i++)
    {
    printf("enter elements:");
    scanf("%d", &ptr[i]);
    }
    printf("\n Succesful allocated using malloc");
    printf("\n the no of elements are:");
    for(i=0; i<n; i++) {
```

```
printf("%d", ptr[i]);
}
free(ptr);
printf("memory freed");
ptr = (int *) calloc(n, sizeof(int));
for(i=0; i<n; i++)
{
  printf("%d", ptr
  printf("\n enter elements %d: ", n);
  scanf("%d", &ptr[i]);
}
printf("\n Successful allocated using calloc");
printf("\n enter no of elements for realloc:");
scanf("%d", &m);
ptr = (int *) realloc(ptr, m * sizeof(int));
printf("\n successful reallocated memory");
printf("\n enter more elements:");
for(i=n; i<m; i++)
{
Printf("\n enter elements:");
scanf("%d", &ptr[i]);
}
printf("elements are:");
for(i=0; i<m; i++)
{
  printf("%d,", ptr[i]);
}
}
return 0;
}
```

output

```
enter no of elements : 3
enter elements : 1
2
3
Succesfull allocated using malloc
the no of elements are : 1, 2, 3, memory freed
enter elements 3 : 1
2
3
Succesful allocated using calloc
enter no of elements for realloc : 5
Succesfull reallocated memory
enter more elements :
enter elements : 4
5
the no of elements are : 1, 2, 3, 4, 5,
```

(3) Stack operations push, pop, display.

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

void push();
void pop();
void display();

int main()
{
 int top = -1, st[size];
 int choice;
```

```
while(1)
{
printf("\n perform operations on stack:");
printf("\n 1. push \n2. pop \n3. display \n4. END");
printf("\n \n enter choice:");
scanf("%d", &choice);

switch (choice)
{
case 1:
   push();
   break;
case 2:
    pop();
    break;
case 3:
     display()
     break;
case 4:
    exit(0);
default:
   printf("\n invalid choice!!");
}
}
return 0; }
void push()
{
  int x;
  if (top == size - 1)
  {
   printf("\n overflow!!");
  }
  else
  {
```

```
printf("\n enter the element to be added onto stack : ");
scanf("%d",&x);
top = top + 1;
st[top] = x;
}
}
void pop() {
 if (top == -1)
 {
 printf("\n underflow !! ");
 }
 else
 {
 printf("\n poped element : %d ", st[top]);
    top = top - 1;
 }
}
void display()
{
 if (top == -1)
 {
 printf("\n underflow !! ");
 }
 else
 {
 printf("\n elements presed in stack : \n");
 for (int i = top; i >= 0; --i)
 printf("%d\n", st[i]);
 }
}
```

output

1. push
2. pop
3. display
4. end

enter choice = 1

enter element to be added onto stack : 5

1. Push
2. pop
3. display
4. end

enter choice = 1

enter element to be added onto stack : 4

1. push
2. pop
3. display
4. end

enter choice : 2

popped element : 4

1. push
2. pop
3. display
4. end

enter choice : 3

element present in stack = 5

1. push
2. pop
3. display
4. end

enter choice = 4.

Sneha
21/12/23

28/12/23

week 3

① infix to postfix

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#Define size 100
char stack[size];
int top = -1;

char push (char x) {
 if ( top = size -1) {
        printf("stack is full \n");
}
else {
      top++;
   stack[top] = x;
   }
      }
char pop ()
{
 char item;
```

```
if (top == -1) {
printf("Stack underflow\n");
 exit(1);
}
else {
 item = stack[top];
 top--;
 return item;
 }
}
int precedence (char symbol) {
 if (symbol == '^') {
    return 3;
}
else if (symbol == '*' || symbol == '-') {
   return 2;
}
else if (symbol == '+' || symbol == '-') {
   return 1;
}
else
   return 0;
        }
int isoperator (char symbol) {
  if (symbol == '^' || symbol == '*' || symbol == '/' || symb
     || symbol == '-') {
     return 1;
     }
     else
        return 0;
}
void infixtopostfix (char infix[], char postfix[]) {
   int i=0, j=0;
   char symbol, temp;
```

```
push('#');

while(infix[i] != '\0'){
    Symbol = infix[i];
    if (Symbol == '('){
        push(Symbol);
    }
else if (isalnum(symbol)){
        postfix[j++] = temp;
    }
    push(Symbol);
}
i++;
}
while(stack[top] != '#'){
        temp = pop();
    postfix[j++] = temp;
    }
    postfix[j] = '\0';
}

int main(){
 char infix[size], postfix[size];
printf("enter infix exp: ");
gets(infix);

infixtopostfix(infix, postfix);

printf("postfix exp: %s\n", postfix);

return 0;
}
```

output

enter infix exp : 2*3+1-3/2^1
postfix exp : 23*1+321^/-

(2) postfix evaluation

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

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

void push(int item){
 if (top == size-1){
    printf("stack overflow\n");
     exit(1);
  }
  else{
     top++;
     stack[top] = item;
  }
}
int pop(){
   int item;
   if (top == -1){
    printf("stack underflow\n");
     exit(1);
    }
    else{
      item = stack[top];
      top--;
      return item;}
```

```
}

int compute (char operator, int operand1, int operand2) {

switch (operator) {
  case '+';
     return operand1 + operand2;
case '-';
   return operand1 - operand2;
case '*';
   return operand2 * operand2;
case '/';
   return operand1 / operand2;
    if (operand2 != 0) {
       return operand1 / operand2;
    }
    else {
      printf ("error: division by zero\n");
       exit (1);
    }
default:
    printf ("invalid expression.\n");
    exit (1);
 }
}
int evaluate postfix (char exp[]) {
   for (int i = 0; exp[i] != '\0'; i++) {
     if (isdigit (exp[i])) {
       push (exp[i] - '0');
     }
else if (exp[i] == '+' || exp[i] == '-' || exp[i] == '*' || exp[i] == '/')
{
 int op2 = pop();
 int op1 = pop();
```

```
int result = compute (exp[i], op1, op2);
  push(result);
  }
}

return stack[top];
}

int main() {
  char postfixexpression[size];
printf("enter postfix expression: ");
  gets(postfixexpression);
  int result = evaluatepostfix(postfixexpression);

printf("result = %d\n", result);

return 0;
}
```

output

enter postfix expression : 23*1+321^/-

result : 1

Stack
2 3 | 1 2 | 2 3

week-3

circular queue

```
#include <stdio.h>
#define size 5
int front = -1;
int rear = -1;
int a[size];
void insert();
void del();
void display();
int main(){
int ch;
while(1){
printf("enter ch 1.insert \n 2.delete \n 3.display \n 4.exit \n");
scanf("%d",&ch);
switch(ch){
case 1: insert();
        break;
case 2: del();
        break;
case 3: display();
        break;
case 4: printf("thank you");
        exit(0);
default: printf("invalid");
    }
  }
}
void insert(){
if(rear==size-1){
    printf("error stack is full");
        return;
}
else{
      int ith;
```

```
printf("enter element");
scanf("%d", &item);
rear = (rear+1) % size;
cq[rear] = item;
}
}
void del() {
    if (front == -1) {
        printf("underflow");
        return;
    }
else {
    int item;
    item = cq[front];
    front = (front+1) % size;
    printf("removed element: %d", item);
    }
    }
void display() {
    if (front == -1) {
        printf("underflow");
        return;
    }
    else {
        for (int i = front; i != rear;) {
            printf("%d\n", cq[i]);
            i = (i+1) % size;
        }
}
}
```

output

enter choice 1. insert
2. delete
3. display
4. exit
1
enter elment 2
enter choice 1. insert
2. delete
3. display
4. exit
4
Thank you.

11/1/24 week

Singly linked list

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

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

Node *head = NULL;

void push();    // insert at beginning
void append();  // insert at last
void insert();  // insert at given position
void display();

int main() {
  int choice;
  while (1) {
    printf("1. insert at beging\n");
    printf("2. insert at end\n");
    printf("3. insert at position\n");
    printf("4. Display\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 pushll {
    Node * link = (Node *) malloc (sizeof (Node));
    int new_data;
    printf ("enter data in new node");
    Scanf ("%d", &new_data);
    link -> data = new_data;
    link -> next = head;
    head = link;
}

void atend() {
Node * link = (Node * ) malloc (sizof (Node));
  int new_data;
  printf ("enter data in the new node: ");
  Scanf ("%d", &(new_data);
  link -> data = new_data;
  link -> next = NULL;
  if (head == NULL) {
      head = link;
      return;
    }
    Node * link1 = head;
    while (link1 -> next != NULL) {
        link1 = link1 -> next;
    }
    link1 -> next = link;
}
void insert () {
  Node * link = (Node *) malloc (sizeof (Node));
```

```
int new_data, pos;
printf("enter data in the new node: ");
scanf("%d", &pos);
link -> data = new_data;
link -> next = NULL;
if (pos == 0) {
    link -> next = head;
    head = link;
    return;
} while (--pos) { link1 = link1 -> next; }
Node * link2 = link1 -> next;
link -> next = link2;
link1 -> next = link;
}

void display () {
    Node * link1 = head;
    while (link1 != NULL) {
        printf("%d" -> ", link1 -> data);
        link1 = link1 -> next;
    }
    printf("NULL");
}
```

output:

1. insert at beginning
2. insert at end
3. insert at position
4. display
5. exit

enter choice: 1
enter data in the new node: 5
enter choice: 2
enter data in the new node: 6

Page

enter choice: 3
enter data in the new node: 1
enter position of the new node: 2

enter choice: 4
5 → 6 → 1 → NULL

enter choice: 6
program exited.

Sneha
11/1/24

18/1/24

week 4

Singly linked list

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

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

Node * head = NULL;

void pop(); // delete from beginning
void end_delete();
void delete_pos();
void display();
void append();

int main() {
    int ch;
while(1){ printf("1. insert at end \n 2. delete from beginning \n 3. delete from end\n
        4. delete at position \n 5. Display \n 6. Exit \n");
printf("enter choice");
scanf("%d", &ch);
switch(ch){
    case 1:
        append(); break;
    case 2:
        pop();
        break;
    case 3:
        end_delete();
        break;
```

```
case 4:
    delete_pos();
    break;
case 5:
    display();
    break;
default:
    printf("exiting");
    exit(0); return 0;
  }
}
}

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

8/1/24

```
head = head -> next;
}

void end_delete() {
    if (head->next==NULL) {
        head
 if (head == NULL) {
   printf("List empty");
   return;
 }
 if (head -> next == NULL) {
     head = NULL;
     return;
  }
 Node* lip = head;
Node* lip1 = head;
lip = lip -> next;
while (lip -> next != NULL) {
    lip = lip -> next;
   lip1 = lip1 -> next;
 }
lip1 -> next = NULL;
}
void delete_pos() {
    int pos;
 printf("enter position");
 scanf("%d", &pos);
Node* lip = head;
while (--pos) {
 lip = lip -> next;
 if (lip -> next = NULL) {
     printf("not enough elements");
  }
}
```

```
Node* tmp1 = tmp;
tmp1 = tmp1 -> next;
if (tmp1 -> next == NULL) {
printf ("not enough elements");
}
tmp1 = tmp1 -> next;
tmp -> next = tmp1;
}

void display() {
Node* tmp1 = head;
while (tmp1 != NULL) {
printf ("%d -> ", tmp -> data);
tmp1 = tmp1 -> next;
}
printf ("NULL \n");
}
```

output 1. insert at end
2. delete from beginning
3. delete from end
4. delete from position
5. Display
6. exit

enter choice: 1
enter new data: 1
enter choice: 1
enter new data: 2
enter choice: 1
enter new data: 3
enter choice: 1
enter new data: 4
enter choice: 2
enter choice: 5
2 -> 3 -> 4 -> NULL

// 1 -> 2 -> 3 -> 4 -> NULL
} deletion from beginning

18/1/24

enter choice : 3 } deletion from end
enter choice : 5
2 → 3 → NULL
enter choice : 1
enter new-data : 5 // 2 → 3 → 5 → NULL
enter choice : 4
enter position : 2 } deletion at position
enter choice : 5
2 → 5 → NULL

leetcode : 155

(Q) Implement the minStack class

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

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

minStack * minStackCreate() {
minStack * stack = (minStack*) malloc (sizeof (minStack));
stack -> top = -1;
return stack;
}

void minStackPush (minStack * obj, int val) {
    if (obj -> top == max-1) {
    printf("Stack Full\n");
    return;
    }
obj -> st[++obj -> top] = val;
```

```
if (obj->top->0)
{
  if (obj->min[obj->top-1] <= val)
    obj->min[obj->top] = obj->min[obj->top-1];
  else
    obj->min[obj->top] = val;
}
else
  obj->min[obj->top] = val;
}
void minStackpop (minStack* obj) {
 if (obj->top == -1)
 {
  printf("Stack empty\n");
    return;
 }
else {
    obj->top -= 1;
  }
}
int minstack Top (minstack* obj) {
   if (obj->top == -1) {
    printf("Stack empty\n");
      return -1;
   }
  return obj->st[obj->top];
}
int minStack GetMin (minStack* obj) {
   if (obj->top == 1) {
printf("min stack empty\n");
      return -1;
    }
    return obj->min[obj->top];
  }
```

18/1/24

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

Sneha
18/1/24

Date
Page

week 5

① Operations on Singly linked list

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

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

struct Node* head1 = NULL;
struct Node* head2 = NULL;

void insertATEnd (struct Node* *head) {
struct Node* newnode = (struct Node*) malloc (sizeof (struct node));
int newval;
Scanf ("%d", &newVal);
newNode -> data = newVal;
newNode -> next = NULL;
if (*head == NULL) {
        *head = newNode;
        return;
    }
Struct Node* tail = *head;
while (tail -> next != NULL) {
    tail = tail -> next;
}
tail -> next = newNode;
}

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

```
printf("NULL");
}

void SortList (struct Node** head){

 Struct Node* current = *head;
 Struct Node* temp = NULL;
 while (current != NULL){
     temp = current -> next;
 while (temp != NULL){
  if (current -> data > temp -> data){
      int swap = current -> data;
         current -> data = temp -> data;
           temp -> data = swap;
       }
       temp = temp -> next;
      }
     current = current -> next;
    }
}

Void reverseList (Struct Node** head){
   Struct Node* prev = Null;
   Struct Node* current = *head;
   Struct Node* next = NULL;

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

```
int main() {

    int l1, l2;

printf("enter length of first linked list : ");
Scanf("%d", &l1);

printf("enter length of first linked list : ");
Scanf("%d", &l1);

printf("enter length of second linked list : ");
Scanf("%d", &l2);

printf("enter first linked list : ");
for(int i=0; i<l1; i++){
    insertATEND(&head1);
}
for(int i=0; i<l2; i++){
    insertATEND(&head2);
}

printf("first linked list : ");
display(head1);

printf("second linked list : ");
display(head2);

sortList(&head1);
printf("\n sorted linked list : ");
display(head1);

reverseList(&head2);
printf("\n Reversed linked list : ");
display(head2);
```

```
concatenate (head1, head2);
printf ("\n Concatenated linked list : ");
display (head1);

return 0;
}
```

output :

```
enter length for l1 : 3
enter length for l2 : 3

enter first linked list : 1
4
5
enter second linked list : 7
8
9
First list : 1 → 4 → 5 → NULL
Sec list : 7 → 8 → 9 → NULL
Sorted list : 1 → 4 → 5 → NULL
Reverse list : 9 → 8 → 7 → NULL
Concatenated list : 1 → 4 → 5 → 9 → 8 → 7 → NULL
```

classmate
Date
Page

② Linked list as Stack

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

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

struct Node* top = NULL;

void push (int element) {

struct Node* newNode = (struct Node*) malloc (sizeof (struct Node));

newNode -> data = element;
newNode -> next = top;
 top = newNode;
}

void pop () {

 if (top == NULL) {
printf ("Stack is empty \n");
 return;
}
struct Node* temp = top;
printf ("Stack : \n");
while (temp != NULL) {
printf ("%d", temp -> data);
    temp = temp -> next;
 }
```

```
temp = top -> next;
top = temp;
```

```
printf("\n");
}

int main() {
 int ch;
while (1) {
 printf("1. push\n 2. pop\n 3. Display\n 4. Exit\n");
printf("enter choice :");
Scanf ("%d", &ch);
Switch (ch) {
 case 1: { int ele;
          printf("enter element:");
          Scanf ("%d", &ele);
           push (ele);
           break; }
case 2 : pop();
          break;
case 3   display();
          break;
default : return 0;
 }
}
return 0;
}
```

output

~~enter choice~~

1. push
2. pop
3. display
4. exit.

enter choice 1

enter element : 1

enter choice 1

enter elmnt : 2

enter choice: 3
1 2
enter choice: 2
enter choice: 3
1
enter choice: 4

(3) Singly linked list as queues.

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

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

struct Node* front = NULL;
struct Node* rear = NULL;

void enqueue (int element) {

struct Node* newNode = (struct Node*) malloc (sizeof (struct Node));

newNode -> data = element;
newNode -> next = NULL;

if (rear == NULL) {
   front = rear = newNode;
   return;
  }
rear -> next = newNode;
   rear = newNode;
}
```

```
void dequeue() {
  if (front == NULL) {
      printf("Queue is empty \n");
      return;
  }
  struct Node* temp = front;
  front = front -> next;

  if (front == NULL)
      rear = NULL;

  free(temp);
}

void display() {
  if (front == NULL) {
  printf("Queue is empty.\n");
    return;
  }
  struct Node* temp = front;
  printf("Queue: \n");
while (temp != NULL) {
printf("%d ", temp -> data);
  temp = temp -> next;
  }
  printf("\n");
}

int main() {
  int ch;
  while (1) {
      printf("1. Enqueue \n2. Dequeue \n3. display \n4. Exit\n");
      printf("Enter choice");
      scanf("%d", &ch);
```

```
Switch (ch) {
  case 1: { int ele;
            printf ("Enter Element :");
            scanf ("%d", &ele);
            enqueue (ele);
            break; }
  case 2: dequeue ();
          break;
  case 3: display ();
          break;
  default: return 0;
  }
}
return 0;
}
```

Output

1. enqueue
2. Dequeue
3- display
4. Exit
enter choice : 1
enter element : 1
enter choice : 1
enter element : 2
enter choice : 1
enter element : 3
enter choice : 1
enter element : 4
enter choice : 3
Queue:
1 2 3 4
enter choice : 2
enter choice : 3
Queue:

5/1/24

week #6

Q. Operations on doubly linked list

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

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

struct Node* createNode() {

    struct Node* newNode = (struct Node*) malloc (sizeof(
int data;
printf("enter data in node");
scanf("%d", &data);
newNode -> data = data;
newNode -> prev = NULL;
newNode -> next = NULL;
return newNode;
}

void insertNode(struct Node** head) {
    int pos;
    printf("enter position of new node");
    scanf("%d", &pos);
    struct Node* newNode = createNode();
    struct Node* temp = (*head);
    while(--pos) {
        if (temp -> next != NULL) {
            temp = temp -> next;
        else {
            printf("list too short");
            return; } }
```

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

void deleteNode (struct Node* *head) {
    int data;
    printf("enter data in node to be deleted");
    scanf("%d", &data);
    struct Node* current = *head;
    while (current != NULL) {
        if (current -> data == data) {
            if (current -> prev != NULL) {
                current -> prev -> next = current -> next;
            } else {
                *head = current -> next;
            }
            if (current -> next != NULL) {
                current -> next -> prev = current -> prev;
            }
            free (current);
            return;
        }
        current = current -> next;
    }
```

```
printf("Node with value %d not found", data);
}

void display(struct Node* head) {

    struct Node* current = head;
    printf("Doubly linked list: ");
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    int choice;
    struct Node* head = NULL;
    while (1) {
        printf("1. Create a list\n 2. insert a node\n 3. Delete a
        node\n 4. Display\n 5. Exit\n enter choice: ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            head = createNode();
            break;
        case 2:
            insertNode(&head);
            break;
        case 3:
            deleteNode(&head);
            break;
        case 4:
            display(head);
            break;
```

```
default:
    printf("exiting program");
    return 0;
}
}
}
```

output:

1. Create list
2. insert a node
3. Delete a node
4. Display
5. Exit

enter choice: 1
enter data in node: 2
enter choice: 2
enter position of new node: 3
enter data in node: 4

enter choice: 4
Doubly linked list: 4 → 2 → Null

enter choice: 3
enter data in node to delete: 2

enter choice: 4
Doubly linked list: 4 → NULL

enter choice: 5
program exited.

1/2/24

week -8

Binary Tree

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

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

struct Node* insert (struct Node* node, int data){
    if (node == NULL) {
    node = (struct Node* node, int data) {
     if (node == NULL){
 node = (struct Node*) malloc (sizeof (struct Node));
 node -> data = data;
 node -> left = NULL;
 node -> right = NULL;
  return node;
}

if (data < node -> data)
    node -> left = insert (node -> left, data);
else if (data > node -> data)
    node -> right = insert (node -> right, data);

 return node;
}

void inorder (struct Node* root) {
    if (root != NULL) {
        indoor (root -> left);
```

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

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

void display(struct Node* root){
    printf("Inorder traversal:");
    inorder(root);
    printf("\nPreorder traversal:");
    preorder(root);
    printf("\n");
}

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

```
int main ()
{
 struct node* root = NULL;
  int ch;
 printf ("1. insert \n 2. display \n 3. exit");
 while (1)
 {
 printf ("\n enter choice");
 scanf ("%d", &ch);
 switch (ch)
 {
  case 1: {printf ("enter value you want to insert");
           int val;
           scanf ("%d", &val);
           root = insert (root, val);
           break;
          }
case 2: display (root); break;
case 3: printf (" thank you"); exit (1);
  }
}
 return 0;
}
```

output:

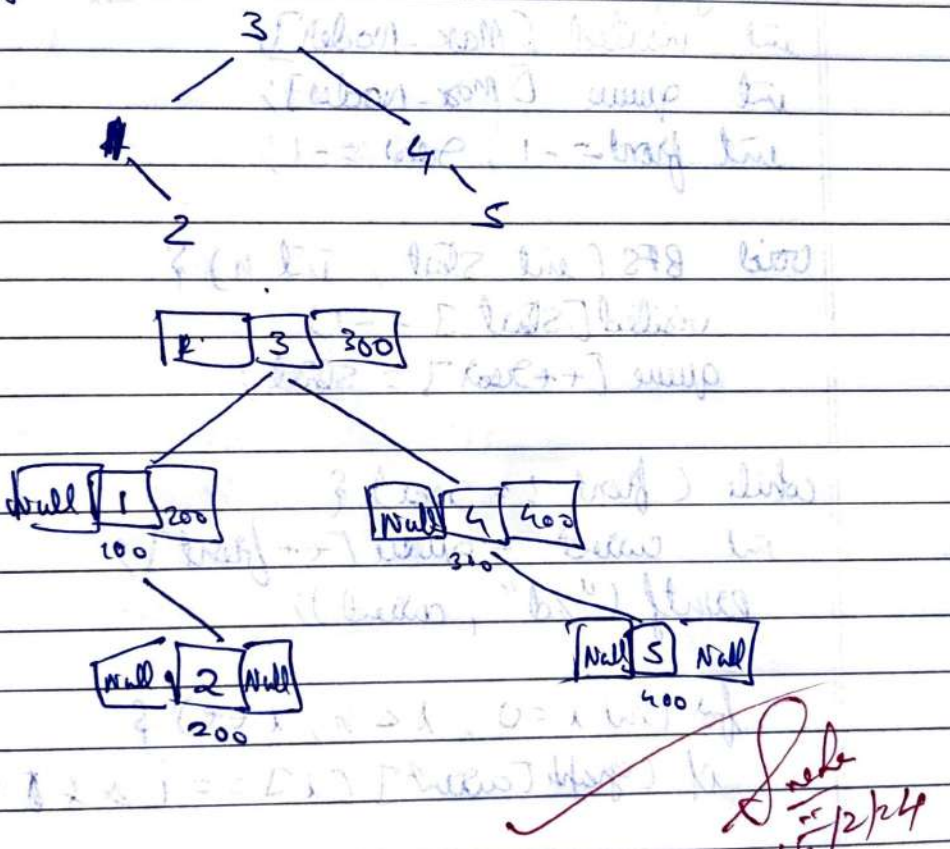
1. insert
2. display
3. exit

enter choice: 1
enter val: 3
enter choice: 1
enter val: 1
enter choice: 1
enter val: 4
enter choice: 1
enter val: 2
enter choice: 1
enter val: 5

enter choice : 2
inorder traversal : 1 2 3 4 5
preorder traversal : 3 1 2 4 5
postorder traversal : 2 1 5 4 3

enter choice : 3
Thank you.

Tracing



Sneha
15/2/24

~~Binary~~, week-9

Breadth first Search

```
#include Max_Nodes 100
#include Max_Nodes 100

int graph[Max_Nodes][Max_Nodes];
int visited[Max_Nodes];
int queue[Max_Nodes];
int front=-1, rear=-1;

void BFS(int start, int n){
    visited[start] = -1;
    queue[++rear] = start;

while (front != rear){
  int current = queue[++front];
   printf("%d", current);

    for(int i=0; i<n; i++){
    if (graph[current][i]==1 && !visited[i]){

        visited[i]=1;
        queue[++rear]=i;
      }
    }
  }
}
```

```
int main(){

int n, m;
printf("enter the number of nodes and edges: ");
scanf("%d %d", &n, &m);

printf("enter the edges: \n");

for(int i=0; i<m; i++){
  int a, b;
  printf("enter initial node to ending:");
  scanf("%d %d", &a, &b);
  graph[a][b]=1;
  graph[b][a]=1;
}
int start;
printf("enter the starting node:");
scanf("%d", &start);

printf("BFS traversal:");
BFS(start, n);

return 0;
}
```

output

```
enter the no of nodes & edges: 5
5
Enter the edges:
0
1
1
2
2
2
```

22/2/24

3
4
4
0
enter starting node: 4
BFS traversal: 4 0 3 1 2

DFS Depth first search

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

#define Max_Nodes 100
#define Max_Nodes 100

int graph[Max_Nodes][Max_Nodes];
int visited[Max_Nodes];

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

int isConnected(int n) {
    DFS(0, n);
    for (int i=0; i<n; i++) {
        if (!visited[i]) {
            return 0;
        }
    }
    return 1;
```

```
}

int main(){
  int n, m;
printf("enter the no of nodes and edges: ");
scanf("%d %d", &n, &m);

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

return 0;
}
```

Outfput

```
enter no of edges nodes & edges: 5
5
enter the edges:
0
1
1
2
2
3
3
```

22/2/24

4
0
The graph is connected.

Lab-5 leetcode

Linked list

```
Struct ListNode* reverseBetween (struct ListNode*
Sleart, int a, int b) {
  a=1;
  b=1;
Struct ListNode* node1 = NULL, *node a = Null;
*ptr = Sleart, *node2 = Null, node b* = NULL;
  int c=0;
while (ptr != NULL){
   if (c==a-1){
      node b = ptr;
  else if (c==a)
      node1 = ptr;
  else if (c==b)
      node2 = ptr;
  else if (c==b+1) {
      node a = ptr; break; }
    c+=1;
   ptr = ptr -> next; }

Struct ListNode *pre = node a, *leif;
      ptr = Sleart;
       e=0;
```

```
while (ptr != NULL) {
  if (c >= a && c < b) {
    [illegible] = ptr -> [illegible];
    ptr -> next = pre;
    pre = ptr;
    ptr = [illegible]; }
else if (c == b) {
    ptr -> next = pre;
      if (a == 0)
        start = ptr;
    else
      node b -> next = ptr;
    break;
  }
  else
    ptr = ptr -> next;
    c++; } return start; }
```

Input

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

left = 2

right = 4

output

[1, 4, 3, 2, 5]

Leetcode - Lab-7

Singly linked list

```
Struct ListNode ** SplitListToparts ( struct ListNode **
head, int K, int *returnSize ) {
struct ListNode* current = head;
  int length = 0;
while( current) {
    length ++;
    current = current -> next; }
  int part_size = length / K;
  int extra_nodes = length % K;

  Struct ListNode **result = (Struct ListNode**)
    malloc (K* size of ( struct ListNode*));
      current = head;
for ( int i=0; i<K; i++) {
   Struct ListNode* part_head = current;
int part_length = part_size + ( i < extranodes ? 1:0);

for ( int j = 0; j < part_length-1 && current ; j++){
      current = current -> next; }

if (current) {
Struct ListNode** next_node = current -> next;
  current -> next = NULL;
   result [i] = part_head;
   current = next_node; }
else { result [i] = Null; }
}
*returnSize = K
  return result; }
```

Output

case 1

head = [1, 2, 3]

k = 5

output = [[1], [2], [3], [8], [1]]

Leetcode - Lab - 8

Rotate LinkedList

```
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;
if ( 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; }
```

output

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

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

Lab - 9 hackerrank

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

output
3
2 3
-1 -1
-1 -1
2

output

3

2 3

-1 -1 => 3 1 2

-1 -1 2 1 3

2

Lab-10
Hashing

```
#include <stdio.h>

#define Table_Size 10

int hash_table[table_size] = {0};

void insert (int key) {
  int i = 0;
  int hkey = key % Table_Size;
  int index = hkey;

while (hash_table[index] != 0) {
   i++;
  index = (hkey + i) % table_Size;
  if (i == Table_size) {
    printf ("Hash table is full\n");
     return;
   }
  }
  hash_table[index] = key;
}

Void search (int key) {
   int i = 0;
  int hkey = key % Table_Size;
  int index = hkey;

while (hash_table[index] != key) {
   i++;
  index = (hkey + i) % Table_Size;
```

```
if (i == Table_size) {
    printf ("Element not found in hash table: \n");
    return;
  }
}
printf ("element found at index %d \n", index);
}

int main() {
  int choice;
  int key;

while (1) {
  Printf ("1. Insert \n 2. Search \n 3. Exit \n");
printf ("Enter choice: ");
Scanf ("%d", &choice);
switch (choice) {
case 1: printf ("enter key: ");
        Scanf ("%d", &key);
        insert (key);
        break;
case 2: printf ("enter key: ");
        Scanf ("%d", &key);
        Search (key);
        break;
case 3: printf ("Thank you");
        ~~break;~~ exit(1);
~~case 4:~~
default: printf ("invalid");
         return 0;
  }
}
return 0;
}
```

29/2/24.