S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	21/12/23	swapping two variables using dynamic memory		
	27/12/23	Stack Implementation		
2	11/01/24	Infix to postfix expression		
3	11/01/24	Queue Implementation Circular Queue Implementation		
4	11/01/24	Singly linked list Implementation		
5	18/01/24	Linked list Operations Leetcode - Minstack		
6	25/01/24	Linked list: Sort, Reverse, Concat linked list - Stacks and Queue Operations		
7	01/02/24	Doubly linked list Leetcode - Reversing linked list		
8	15/02/24	Construct binary search tree Leetcode - Split Linked list in parts		
9	22/02/24	Traverse using BFS graph is connected - DFS Leetcode - Rotate list		
10	29/02/24	Hashing Hackerrank - Split		

Practice Questions

Output 1:

1) enter 1 - to create a account 2 - to withdraw
3 - to deposit 4 - to check balance 5 - exit
enter the name : Rushila. V
enter the age : 19
enter 1 - to create a account 2 - to withdraw
3 to deposit 4 - to check balance 5 - to exit
3
enter the amount to deposit : 1000 0
enter 1 - to create a account 2 : to withdraw
3 to deposit 4 - to check balance 5 : exit
3
enter the amount to deposit : 10000
enter 1 - to create a account 2 : to withdraw
3 - to deposit 4 - to check balance 5 : exit
5

Output 2:

2) Enter the number of strings : 4
enter strings : Hello
enter string 2 : Bye
enter string 3 : World
enter string 4 : Hi
Sorted strings
Bye
Hello
Hi
World

Output 3:

3) read n : 2
read n : 2
read the elements :

3
4
enter the key element : 2
element found
output 4:

4) enter a larger string: Useful
enter a substring to search for: Use
substring found
output 9:

5) Read n : 5
Enter the elements of the array
1
2
2
6
4
enter the number to find the last occurence : 2
last occurence of 2 is at 3
Output 6:

6) read n : 5
read the elements
1
6
8
9
10
enter the key element : 3
element found at 3

Output 7:

7) enter n: 5
read the elements
10
11
12
13
14
enter the key element: 13
element found at 4

Output 8:

8) Read n: 5
enter the elements of the array
1
9
22
62
53
62 maximum no in array
1 minimum no in array

21/12/23

```
#include<stdio.h>
void swapref(int *a, int *b);
void main()
{
  int a=10;
  int b=45;
  printf("value of a and b before swapping =%d%d\n",a,b);
  swapref(&a,&b);
  printf("swapping of two values after call by reference: %d %d\n",a,b);
}
void swapref(int *a, int *b)
{
  int temp;
  temp=*a;
  *a=*b;
  *b=temp;
}
```

Output:
value of a and b before swapping = 10 45
the values of and b after swapping = 45 10

21/12/23

```
#include<s
#include<s
void mai
{
  int *p, *
  int n;
  int i;
  printf("
  scanf(
  p=(int
  printf
  for(i=
  {
    scan
  }
  a=(in
  print
  for(
  {
    scan
  }
  p=
  pri
  for
  }
  }
  fo
  fo
}
```

21/12/23

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
 int *p, *q, i;
 int n;
 int i;
 printf("read n");
 scanf("%d", &n);
 p=(int *) malloc(n*sizeof(int));
 printf("enter %d elements", n);
 for(i=0; i<n; i++)
 {
  scanf("%d", p+i);
 }
 q=(int *) calloc(n, sizeof(int));
 printf("Enter %d elements", n);
 for(i=0; i<n; i++)
 {
 scanf("%d", q+i);
 }
 p=realloc(p, 7*sizeof(int));
 printf("Enter seven elements");
 for(i=0; i<7; i++)
 {
  scanf("%d", p+i);
 }
 free(p);
 free(q);
}
```

Output:

read n: 3
enter 3 elements: 2
10
3
4
enter 3 elements:
14
5
7
enter seven elements
6
2
8
9
3
4

21/12/23

#include<s
#include<s
#define m
int top=
int s[ma
void push
{
if (top
{
printf
}
else
{
top=
s[to
}
}
void po
{
int
if (to
printf
}
else
{
val
top
pri
}
}

21/12/23

```
#include<stdio.h>
#include<stdlib.h>
#define max 5
int top=-1;
int s[max];
void push(int value);
{
  if(top==max-1)
  {
    printf("stack overflow can't push");
  }
  else
  {
    top=top++;
    s[top]=value;
  }
}
void pop()
{
  int value;
  if(top==-1)
  {
    printf("stack underflow");
  }
  else
  {
    value=s[top];
    top=top-1;
    printf("%d is popped",value);
  }
}
```

```
void display()
{
  if (top==-1)
  printf("stack is full underflow");
  clx
  printf("stack elements are:");
  for(int i=0; i<=top; i++)
  printf("%d\t",s[i]);
}
void display main()
{
  int value;
  int num;
  printf("Enter a number");
  scanf("%d",&num); push(num);
  printf("Enter a number");
  scanf("%d",&num);
  push(num).
  printf("Enter a number")
  scanf("%d",&num);
  push(num).
  display();
  pop();
  pop();
  pop();
  display();
}
```

Output:

Enter a n
Enter a n
Enter a n
Stack is
Stack el
20 is p
30 is p
10 is p
Stack el
Stack

Output:

Enter a no: 10
Enter a no: 20
Enter a no: 30
Stack is overflow can't push
Stack elements are 10 20 30
20 is popped
30 is popped
10 is popped
Stack elements are: 10 10 is popped
Stack is underflow: can't pop

21/12/23

23/12/23

Evaluation of infix to postfix

(1)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 100
char stack[max];
char infix[max], postfix[max];
int top=-1;
void push(char);
char pop();
int isEmpty();
void InToPost();
void print();
int precedence(char);
int main()
{
  printf("Enter the infix expression");
  gets(infix);
  InToPost();
  print();
  return 0;
}
void InToPost()
{
  int i, j=0;
  char symbol, next;
  for(i=0; i<strlen(infix); i++)
  {
    symbol=infix[i];
    switch(symbol)
    {
      case '(':
        push(symbol);
        break;
```

```
case ')':
  while((next=pop())!='(')
  {
    postfix[j++]=next;
  }
  break;
case '+':
case '-':
case '*':
case '/':
case '^':
  while(!isEmpty() && prec(stack[top])>=prec(symbol))
  {
    postfix[j++]=pop();
  }
  push(symbol);
  break;
default:
  postfix[j++]=symbol;
}
}
while(!isEmpty())
{
  postfix[j++]=pop();
}
postfix[j]='\0';
}
int precedence(char symbol)
{
  switch(symbol)
  {
    case '^':
      return 3;
    case '*':
    case '/':
      return 2;
    case '+':
    case '-':
      return 1;
    case '(':
      return 0;
    default:
      return -1;
  }
}
```

```
case ')':
    while ((next=pop()) != '(')
    postfix[j++] = next;
    break;
case '+':
case '-':
case '*':
case '/':
case '^':
    while (!isEmpty() && precedence(stack[top])
           >= precedence(symbol))
    postfix[j++] = pop();
    push(symbol);
    break;
default:
    postfix[j++] = symbol;
}
}
while (!isEmpty())
postfix[j++] = pop();
postfix[j] = '\0';
}

int precedence(char symbol)
{
  switch(symbol)
  {
    case '^':
        return 3;
    case '/':
    case '*':
        return 2;
    case '+':
    case '-':
        return 1;
    default:
        return 0;
  }
}
```

```
void fprint()
{
int i=0;
fprintf("the equivalent postfix expression is");
while(postfix[i])
{
fprintf("%c",postfix[i++]);
}
fprintf("\n");
}
void fpush(char c)
{
if(top==MAX-1)
{
fprintf("stack overflow");
return;
}
top++;
stack[top]=c;
}
char fpop()
{
char c;
if(top==-1)
{
fprintf("stack underflow");
exit(1);
}
c=stack[top];
top=top-1;
return c;
}
int isEmpty()
{
if(top==-1)
return 1;
else
return 0;
}
```

Output:
Enter the infix
the equivalent

```
int isEmpty()
{
  if (top == -1)
    return 1;
  else
    return 0;
}
```

Output:
Enter the infix expression: (a*b)+(c*d)
The equivalent postfix expression is: ab*cd*+

ALP 11/11/24

Queue

②
```
#include<stdio.h>
#include<stdlib.h>
#define max 6
int Q[MAX];
int front=-1;
int rear=-1;
void insert(int x){
  if(rear==MAX-1)
  {
    printf("Queue overflow");
    exit(1);
  }
  if(front==-1){
    front=rear=0;
  }
  else
  {
    rear++;
  }
  Q[rear]=x;
}
void delete(){
  if(front==-1)
  {
    printf("Queue underflow");
    exit(0);
  }
  printf("deleted element is %d\n",Q[front]);
  front++;
  if(front>rear)
  {
    front=rear=-1;
  }
}
```

```
void display()
{
 if (front == -1)
 {
  printf("Queue underflow");
  exit(1);
 }
 int i;
 printf("front to rear: ");
 for(i=front; i<=rear; i++)
 {
  printf("%d", Q[i]);
 }
 printf("\n");
}
void main(){
 int choice;
 int x;
 do{
  printf("Queue Operations");
  printf("1. insertion");
  printf("2. deletion\n");
  printf("3 display\n");
  printf("4. exit\n");
  printf("enter the choice");
  scanf("%d", &choice);
  switch(choice)
 {
  case 1:
   printf("Enter the element to be inserted");
   scanf("%d", &x);
   insert(x);
   break;
  case 2: delete();
          break;
```

```
case 3: display();
        break;
case 4: exit(1);
default: printf("Invalid choice try again");
}while(choice!=4);
}
```

Output:

Queue operation
1. insertion
2. deletion
3. display
4. exit
enter the choice 1
enter the element to be inserted 34

Queue operation
1. insertion
2. deletion
3. display
4. exit
enter the choice 2
enter the element to be deleted 34

Queue operation
1. insertion
2. deletion
3. display
4. exit
enter the choice 4

Week-4

11/01/24 // Circular queue

```
#include <stdio.h>
#include<stdlib.h>
#define size 50
int queue[size];
int rear = -1;
int front = -1;
int isFull()
{
  if (front == (rear+1)%(size))
  {
    return 0;
  }
  else
  {
    return -1;
  }
}
int isEmpty()
{
  if (front == -1 && rear == -1)
  {
    return 0;
  }
  else
  {
    return -1;
  }
}
void Enqueue(int x)
{
  int item;
  if (isFull() == 0)
  {
    printf("Queue overflow\n");
    return;
```

```
else
{
  if (IsEmpty() == 0)
  {
    front = 0;
    rear = 0;
  }
  else
  {
    rear = (rear+1) % size;
  }
  Q[rear] = x;
}

int Dequeue()
{
  int x;
  if (IsEmpty() == 0)
  {
    printf("Queue underflow\n");
  }
  else
  {
    if (front == rear)
    {
      x = queue[front];
      front = -1;
      rear = -1;
    }
    else
    {
      x = queue[front];
      front = (front+1) % size;
    }
    return x;
```

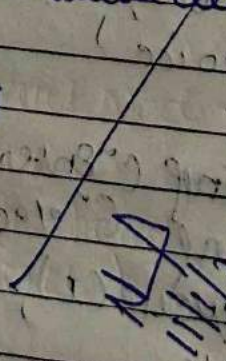
```
void Display()
{
  int i;
  if (IsEmpty
  {
    printf("Q
  }
  else
  {
    printf("
    for (i=
    {
      printf
    }
    printf("
  }
void main
{
  int cho
  while(1
  {
    print
    scanf
    scanf
    switch
    {
      case
      case
```

```
void display()
{
int i;
if (isEmpty() == 0)
{
  printf("Queue is empty\n");
}
else
{
  printf("Queue elements:\n");
  for(i=front; i!=rear; i=(i+1)%size)
  {
    printf("%d\n", queue[i]);
  }
  printf("%d\n", queue[i]);
}
void main()
{
  int choice, x, b;
  while(1);
  {
    printf("1.Enqueue 2.Dequeue 3.display 4.exit\n");
    printf("Enter your choice");
    scanf("%d", &choice);
    switch(choice)
    {
    case 1: printf("Enter the number to be inserted\n");
            scanf("%d", &x);
            enqueue(x);
            break;
    case 2: b = dequeue();
            printf("%d was removed from the
                    queue\n", b);
            break;
```

```
case 3: Display();
        break;
case 4: exit(1);
default: printf("Invalid choice");
    }
}
```

Output

1 Enqueue 2 dequeue 3. display 4. exit
Enter your choice: 1
Enter the number to be inserted
23
1. Enqueue 2. Dequeue 3 display 4. exit
Enter the number to be inserted
45
1 Enqueue 2. Dequeue 3. display 4. exit
Enter the number to be inserted
28 1. Enqueue 2. dequeue 3 display 4. exit
Enter the choice: 2
Enter the no. to be deleted
45
Enter your choice 3. Queue elements
28 28
Enter your choice 4



Week-4

11/01/24

#include <std
#include <st
struct Node
{
int data
struct N
};
struct Nod
{
struct N
(size of
if (nod
{
print
exit(
}
newNo
newNo
return
}
struct N
int val
{
struct
newNo
return
}
struct
{
struc
Afte
if (p
ne

Week-4

11/01/24 // Circular 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));
  if(newNode == NULL)
  {
    fprintf("memory allocation failed\n");
    exit(1);
  }
  newNode -> data = value;
  newNode -> next = NULL;
  return newNode;
}
struct Node * insertAtBeginning(struct Node * head,
int value)
{
  struct Node * newNode = createNode(value);
  newNode -> next = head;
  return newNode;
}
struct Node * insertAtAnyPos(struct Node * head,
                             int value, int pos)
{
  struct Node * newNode = createNode(value);
  struct Node * temp = head;
  if(pos == -1) {
    newNode -> next = head;
    return newNode;
```

```
}
int i;
for(i=1; i<pos-1; i++)
  temp = temp->next;
}
newNode->next = temp->next;
temp->next = newNode;
return head;
}
struct Node * insertAtEnd(struct Node * head,
int value)
{
  struct Node * newNode = createNode(value);
  if(head == NULL)
  {
    return newNode;
  }
  struct Node * temp = head;
  while(temp->next != NULL)
  {
    temp = temp->next;
  }
  temp->next = newNode;
  return head;
}
void displayList(struct Node * head)
{
  struct Node * temp = head;
  while(temp != NULL)
  {
    printf("%d->", temp->data);
    temp = temp->next;
  }
  printf("NULL\n");
```

```
int main()
{
struct node *head = NULL;
int choice, value, pos;
while(1)
{
printf("1. Insert at end\n");
printf("2. Insert at beginning\n");
printf("3. Insert at any position\n");
printf("4. Display list\n");
printf("5. Exit\n");
printf("Enter your choice:");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("Enter a new value to be inserted");
        scanf("%d", &value);
        head = insertAtEnd(head, value);
        break;
case 2: printf("Enter the value to be inserted");
        scanf("%d", &value);
        head = insertAtBeginning(head, value);
        break;
case 3: printf("Enter the value to be inserted");
        scanf("%d", &value);
        printf("Enter the position");
        scanf("%d", &pos);
        head = insertAtAnyPos(head, value, pos);
        break;
case 4: displayList(head);
        break;
case 5: exit(0);
default: printf("Invalid choice");
}
}
return 0;
```

Output

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

Enter your choice: 2
Enter the value to be inserted: 12

Enter your choice: 2
Enter the value to be inserted: 13

Enter your choice: 4
13 → 12 → NULL

Enter your choice: 1
Enter the value to be inserted: 23

Enter your choice: 4
13 → 12 → 23 → NULL

Enter your choice: 3
Enter the value to be inserted: 12
Enter the position to insert: 3

Enter your choice: 4
13 → 12 → 12 → 23 → NULL

18/6/24 Week - 5

#include <stdio
#include <std
struct Node
{
int data;
struct Node
};
struct node
void insert
void begin
void end(
void searc
void displ
int main
{
int cho
while (
{
print
print
print
print
print
print
print
scan
switc
{
case

case

case

18/07/24 Week-5

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
  int data;
  struct Node * next;
};
struct node * head=NULL;
void insert();
void begin();
void end();
void anyfpos();
void display();
int main()
{
  int choice;
  while(1)
  {
    printf("\n1. Insert elements\n");
    printf("2. Delete at the beginning\n");
    printf("3. Delete at the end\n");
    printf("4. Delete at any position\n");
    printf("5. Display\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch(choice)
    {
      case 1: insert();
              break;
      case 2: begin();
              break;
      case 3: end();
              break;
```

```
case 4: atanypos();
        break;
case 5: display();
        break;
case 6: exit(0);
default:
    printf("Invalid choice\n");
}
}
return 0;
}
void invert()
{
  struct Node *ptr, *temp;
  int data;
  ptr = (struct Node *)malloc(sizeof(struct Node));
  if (ptr == NULL)
  {
    printf("memory allocation failed\n");
    exit(1);
  }
  printf("Enter data to insert: ");
  scanf("%d", &data);
  ptr->data = data;
  ptr->next = NULL;
  if (head == NULL)
  {
    head = ptr;
    printf("Node inserted successfully\n");
  }
  else
  {
    temp = head;
    while (temp->next != NULL)
```

```
    {
        temp = temp -> next;
    }
    temp -> next = ptr;
    printf("Node inserted successfully");
    }
}
void begin()
{
    struct Node *ptr;
    if (head == NULL)
    {
        printf("List is empty");
    }
    else
    {
        ptr = head;
        head = ptr -> next;
        free(ptr);
        printf("Node deleted at the beginning\n");
    }
}
void end()
{
    struct Node *ptr, *prev;
    if (head == NULL)
    {
        printf("List is empty\n");
    }
    else
    {
        ptr = head;
        while (ptr -> next != NULL)
        {
            prev = ptr;
            ptr = ptr -> next;
        }
```

```
if(ptr==head)
{
  head=NULL;
}
else
{
  prev->next=NULL;
}
free(ptr);
printf("Node deleted from the end\n");
}
}
void arrayptr()
{
  struct Node *ptr, *prev;
  int loc, i=1;
  printf("Enter the position: ");
  scanf("%d", &loc);
  if(head==NULL)
  {
    printf("list is empty\n");
    return;
  }
  ptr=head;
  if(loc==1)
  {
    head=ptr->next;
    free(ptr);
    printf("Node deleted from position %d\n", loc);
    return;
  }
```

```
while(ptr)
{
  prev=ptr
  ptr=ptr
  i++;
}
if(ptr==
{
  printf("
}
else
{
  prev->
  free(pt
  printf(
}
}
void displ
{
  struct N
  if(head
  {
    printf
  }
  else
  {
    ptr=h
    while
    {
      prev
      ptr
    }
    printf
  }
}
```

```
while(ptr!=NULL && i<loc)
{
  prev=ptr;
  ptr=ptr->next;
  i++;
}
if(ptr==NULL)
{
  printf("Position %d is out of bounds\n",loc);
}
else
{
  prev->next=ptr->next;
  free(ptr);
  printf("Node deleted from position %d\n",loc);
}
}
void display()
{
  struct Node *ptr;
  if(head==NULL)
  {
    printf("list is empty");
  }
  else
  {
    ptr=head;
    while(ptr!=NULL)
    {
      printf("%d",ptr->data);
      ptr=ptr->next;
    }
    printf("\n");
  }
}
```

Output

Menu:
1. Insert elements
2. Delete at the beginning
3. Delete at the end
4. Delete at any position
5. Display
6. Exit
Enter your choice: 1
Enter data to insert: 12

Enter your choice: 1
Enter data to insert: 13

Enter your choice: 1
Enter data to insert: 14

Enter your choice: 1
Enter data to insert: 15

Enter your choice: 5
12 13 14 15

Enter your choice: 2
Node deleted from beginning

Enter your choice: 5
13 14 15

Enter your choice: 3
Node deleted at the end

Enter your choice: 5
13 14

18/11/24

leetcode

```
typedef structure{
 int size;
 int top;
 int *s;
 int *minstack;
}ministack;

minstack *miniStackCreate(){
 minstack *st = (minstack*) malloc (sizeof
   (ministack));
 if (st == NULL)
{
  printf ("Memory allocation failed");
  exit(0);
}
st→size = 5;
st→top = -1;
st→s = (int*) malloc (st→size * sizeof (int));
st→minstack = (int*) malloc (st→size *
          sizeof (int));
if (st→s == NULL)
{
  printf ("memory allocation failed");
  free (st→s);
  free (st→minstack);
  exit (0);
}
return st;
}
```

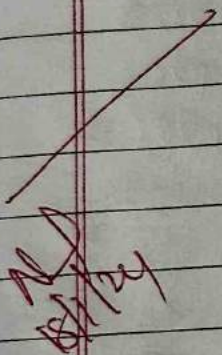
```
void minstackPush(MinStack * obj, int val) {
  if (obj->top == obj->size-1)
{
  printf ("Stack is overflow");
}
else {
  obj->top++;
  obj->s[obj->top]= value;
  if (obj->top==0 || val< obj->minstack
     [(obj->top-1)]){
  obj->minstack[obj->top] = val;
} else{
  obj->minstack[obj->top] = obj->minstack
  [obj->top-1];
}
}
}

void minStackPop(minstack * obj)
{
  int value;
  if (obj->top == -1)
  {
    printf("underflow");
  }
  else
  {
    value=obj->s[obj->top];
    obj->top--;
    printf ("%d is popped in ", value);
  }
}
```

```
int ministackTop(Ministack *obj)
{
  int value = -1;
  if (obj->top == -1)
  {
    printf("underflow\n");
    exit(0);
  }
  else
  {
    value = obj->s[obj->top];
    return value;
  }
}
int ministackTop(Ministack *obj){
  int value = -1;
  if (obj->top == -1)
  {
    printf("underflow\n");
    exit(0);
  }
  else
  {
    value = obj->s[obj->top];
    return value;
  }
}
int ministackGetMin(Ministack *obj){
  if (obj->top == -1)
  {
    printf("underflow\n");
    exit(0);
  }
```

```
else
{
  return o
}
}
void minis
{
  free(o
  free(o
  free(
}
```

```
else
{
 return obj->minstack[obj->top];
}
}
void minStackFree(minStack *obj)
{
   free(obj->s);
   free(obj->minStack);
   free(obj);
}
```



25/01/24 Week-6 : Linked List : Sort, Reverse, Concatenation

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
  int data;
  struct Node * next;
};
struct Node * insertNode(struct Node * head,
int new_data){
  struct Node * new_node = (struct Node *) malloc
                          (sizeof(struct Node));
  new_node -> data = new_data;
  new_node -> next = head;
  return new_node;
}
struct Node * bubbleSort(struct Node * head)
{
  if(head == NULL || head -> next == NULL)
  {
    return head;
  }
int swapped;
struct Node * temp;
struct Node * end = NULL;
do{
  swapped = 0;
  temp = head;
  while (temp -> next != end)
  {
    if (temp -> data > temp -> next -> data)
    {
      int tempData = temp -> data;
      temp -> data = temp -> next -> data;
      temp -> next -> data = tempData;
      swapped = 1;
}
```

```
temp = temp -> next;
}
end = temp;
} while (bisapped);
return head;
}
struct Node * reversedist (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;
  }
  return prev;
}
struct Node * concatdist (struct Node * head1,
                          struct Node * head2)
{
  if (head1 == NULL)
  {
    return head2;
  }
  struct Node * temp = head1;
  while (temp != NULL)
  {
    printf ("%d", temp -> data);
    temp = temp -> next;
  }
  printf ("\n");
}
```

```
int main()
{
  struct Node
  int choice,
  while(1)
  {
    printf("
    printf("
    scanf
    switch
    case 1:
      pri
      sca
      hea
      pre
    case 2:
      pri
      pre
    case 3:

    case 4:
```

```
int main()
{
struct Node * head = NULL;
int choice, data;
while(1)
{
printf("1. Insert node \n 2. Sort linked list \n
3. Reverse linked list \n 4. Concatenate
linked list \n 5. Print linked list \n
6. Exit \n");
printf("Enter your choice");
scanf("%d", &choice);
switch(choice){
case 1:
printf("Enter the data to be inserted:");
scanf("%d", &data);
head = insertNode(head, data);
break;
case 2: head = bubbleSort(head);
printf("linked list sorted \n");
break;
case 3: head = reverselist(head);
printf("Linked list reversed \n");
break;
case 4: printf("Enter the data for the second
linked list");
struct Node * head2 = NULL;
while(1){
scanf("%d", &data);
if(data == -1)
break;
head2 = insertNode(head2, data);
}
head = concatlist(head, head2);
```

printf("Linked lists concatenated\n");
break;
case 5: printlist(head);
break;
case 6: exit(0);
default: printf("Invalid choice\n");
}
}
return 0;
}

Output:
1. InsertNode
2. Sort linked list
3. Reverse linked list
4. Concatenate linked list
5. Print linked list
6. Exit
Enter your choice: 1
Enter the data to insert: 14

Enter your choice: 1
Enter the data to insert: 10

Enter your choice: 1
Enter the data to insert: 15

Enter your choice: 1
Enter the data to insert: 18

Enter your choice: 2
10 14 15
list is sorted
Enter your choice: 5
10 14 15 18

Enter your choice
linked list
Enter your ch
18 15 14
Enter your ch
Enter the da
12 9 8
Enter your
18 15 14
Enter your

Enter your choice: 3
linked list is reversed

Enter your choice: 5
18 15 14 10

Enter your choice: 4
Enter the data for second linked list
12 9 8 7 1

Enter your choice: 5
18 15 14 10 12 9 8 7

Enter your choice: 6

Ended
28/11/24

Stack implementation

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
int data;
struct Node *next;
};
struct Node *createNode(int data)
{
  struct Node *newNode, (struct Node *)
malloc (sizeof(struct Node));
  newNode->data=data;
  newNode->next=NULL;
  return newNode;
}
void push (struct Node **top, int data)
{
  struct Node *newNode = createNode(data);
  newNode->next = *top;
  *top = newNode;
  printf ("%d pushed to the stack \n", data);
}
int pop (struct Node **top) {
  if (*top == NULL)
{
  printf ("stack is empty\n");
  return -1;
}
int poppedvalue = (*top)->data;
struct Node *temp = *top;
*top = (*top)->next;
free (temp);
return poppedvalue;
}
```

```
void displayStack (struct Node * top)
{
  if (top == NULL)
  {
    printf ("Stack is empty\n");
    return;
  }
  printf ("Stack elements: ");
  while (top != NULL)
    printf ("%d", top -> data);
    top = top -> next;
  }
  printf ("\n");
}
int main ()
{
  struct Node * stackTop = NULL;
  int choice, data;
  while (1)
  {
    printf ("1. Push\n 2. Pop\n 3. DisplayStack\n
      4. Exit\n");
    printf ("Enter your choice");
    scanf ("%d", &choice);
    switch (choice)
    {
      case 1: printf ("Enter data to push: ");
      scanf ("%d", &data);
      push (&stackTop, data);
      break;
      case 2: printf ("Popped element: %d\n",
          pop (&stackTop));
          break;
      case 3: displayStack (stackTop);
          break;
```

```
case 4: exit(0);
default: printf("Invalid choice");
}
}
return 0;
}
```

Output:

```
1. Push
2. Pop
3. Display stack
4. Exit
Enter your choice: 1
Enter data to push: 12
12 pushed to the stack

1. Push
2. Pop
3. Display stack
4. Exit
Enter your choice: 1
Enter data to push: 13
13 pushed to the stack

1. Push
2. Pop
3. Display stack
4. Exit
Enter your choice: 1
Enter data to push: 14
14 pushed to the stack
```

1. Push
2. Pop
3. Display stack
4. Exit
Enter your choice: 1 Enter data to push: 13
13 pushed to the stack

1. Push
2. Pop
3. Display stack
4. Exit
Enter your choice: 2
Popped element: 13

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

Queue implementation

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
  int data;
  struct Node * next;
};
struct node * head = NULL;
void enqueue()
{
  struct node * temp;
  temp = (struct node *) malloc(sizeof(struct node));
  printf("Enter value to be inserted: \n");
  scanf("%d", &temp -> data);
  temp -> next = NULL;
  if(head == NULL)
  {
    head = temp;
    return;
  }
  else
  {
    struct node * ptr;
    ptr = head;
    while(ptr -> next != NULL)
    {
      ptr = ptr -> next;
    }
    ptr -> next = temp;
  }
}
```

```
void dequ
{
  struct no
  if(head
  {
    printf
    retur
  }
  else
  {
    ptr =
    head
    print
    free
  }
}
void
{
  if(h
  {
    pri
    ret
  }
  else
  {
    st
    f
    {
    }
  }
}
```

```
void dequeue()
{
  struct node *ptr;
  if(head==NULL)
  {
    printf("Queue is empty\n");
    return;
  }
  else
  {
    ptr=head;
    head=ptr->next;
    printf("value dequeue=%d",ptr->data);
    free(ptr);
  }
}
void display()
{
  if(head==NULL)
  {
    printf("Queue is empty\n");
    return;
  }
  else
  {
    struct node *ptr;
    ptr=head;
    while(ptr!=NULL)
    {
      printf("%d\t",ptr->data);
      ptr=ptr->next;
    }
  }
}
```

```
void main ()
{
int choice;
while(1)
{
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: enqueue();
        break;
case 2: dequeue();
        break;
case 3: display();
        break;
case 4: exit(0);
        break;
}
}
}
```

Output
1.enque
2.deque
3.displa
4.exit
enter yo
enter va

enter yo
enter va

enter yo
enter v

enter yo
enter v

enter
enter

enter
Valu

enter
valu

ente
1

output
1. enqueue
2. dequeue
3. display
4. exit
enter your choice : 1
enter value to be inserted : 12

enter your choice : 1
enter value to be inserted : 54

enter your choice : 1
enter value to be inserted : 19

enter your choice : 1
enter value to be inserted : 20

enter your choice : 1
enter value to be inserted : 30

enter your choice : 3
enter value to be inserted : 12 54 19 20 30

enter your choice : 2
Value dequeue 12

enter your choice : 2
value dequeue : 54

enter your choice : 3
19 20 30

01/02/24

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *prev;
    struct Node *next;
};
struct Node *createNode(int value)
{
    struct Node *newNode = (struct Node *)malloc
                           (sizeof(struct Node));
    if (newNode == NULL)
    {
        fprintf("memory allocation failed\n");
        exit(1);
    }
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
void insertToLeft(struct Node **head, int value
{
    struct Node *newNode = createNode(value);
    if (*head == NULL){
        *head = newNode;
    }
    else
    {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    }
}
```

```
void deleten
{
    if (*hea
    {
        fprintf(
        return
    }
    struct N
    while (cu
    {
        if (cu
        {
            if (c
                cu
        } else
            *he
        }
        if (cu
            cu
        }
        free
        fre
        ret
    }
    curr
}
fre
}
voi
{
```

```
void deleteNode(struct Node **head, int value)
{
  if (*head == NULL)
  {
    printf("List is empty\n");
    return;
  }
  struct Node *current = *head;
  while (current != NULL)
  {
    if (current->data == value)
    {
      if (current->prev != NULL){
        current->prev->next = current->next;
      } else {
        *head = current->next;
      }
      if (current->next != NULL) {
        current->next->prev = current->prev;
      }
      free(current);
      printf("Node with value %d deleted\n", value);
      return;
    }
    current = current->next;
  }
  printf("Node with value %d not found\n", value);
}
void displaylist(struct Node *head)
{
  if (head == NULL)
  {
    printf("list is empty\n");
    return;
  }
```

```
printf("Doubly Linked list:");
while(head != NULL)
{
printf("%d <-> ", head->data);
head = head->next;
}
printf("NULL\n");
}
int main()
{
struct node *head = NULL;
int choice, value;
do {
printf("\n1. Insert to left\n2. Delete by
3. Display\n4. Exit\n");
printf("Enter your choice:");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("Enter the value to insert:");
scanf("%d", &value);
insertToLeft(&head, value);
break;
case 2: printf("Enter the value to delete:");
scanf("%d", &value);
deleteNode(&head, value);
break;
case 3: displayList(head);
break;
case 4: printf("Exiting the program\n");
break;
default: printf("Invalid choice. Please enter
a valid option\n");
}
} while(choice != 4);
return 0;
```

leetcode: Linked

output:

Output:
1 Insert to le
2. Delete by
3. Display
4. Exit
Enter your c
Enter the v

Enter your
Enter the

Enter your
Enter val

Enter you
enter val

Enter yo
positi

Enter
Enter
Node

Enter
Positi

Leetcode: Linked list

output:

Output:
1 Insert to left
2. Delete by value
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 12

Enter your choice: 1
Enter the value to insert: 13

Enter your choice: 1
Enter value to insert: 14

Enter your choice: 1
Enter value to insert: 5

Enter your choice: 3
Doubly linked list: 12 <=> 13 <=> 14 <=> NULL

Enter your choice: 2
Enter the value: 13
Node with value 13 deleted

Enter your choice: 3
Doubly linked list: 12 <=> 14 <=> NULL

leetcode : Reversing linked list

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    if (head == NULL || left == right)
    {
        return head;
    }
    struct ListNode *dummy = (struct ListNode*)malloc(sizeof(struct ListNode));
    dummy->next = head;
    struct ListNode *prev = dummy;
    for (int i = 1; i < left; ++i) {
        prev = prev->next;
    }
    struct ListNode *current = prev->next;
    struct ListNode *next = NULL;
    struct ListNode *tail = current;

    for (int i = left; i <= right; i++)
    {
        struct ListNode *temp = current->next;
        current->next = next;
        next = current;
        current = temp;
    }
    prev->next = next;
    tail->next = current;

    struct ListNode *result = dummy->next;
    free(dummy);
    return result;
```

15/02/24 Week-8 : Bina

```
#include <stdio
#include <stdl
struct node {
    int data;
    struct node
    struct node
};
struct node
    struct no

    node->da
    node->le
    return n
}
structnod
{
    if (roo
    retu
    if (d
    roo
    else
    roo
    retu
}
void
    if
    r
    ino
    fo
    in
}
```

15/02/24 Week-8 : Binary search tree

```
#include<stdio.h>
#include<stdlib.h>
struct node {
  int data;
  struct node *left;
  struct node *right;
};
struct node * NewNode(int data) {
  struct node *node = (struct node *) malloc
                      (sizeof(struct node));
  node->data = data;
  node->left = node->right = NULL;
  return node;
}
structnode *insert(struct node *root, int data)
{
  if (root == NULL)
    return NewNode(data);
  if (data <= root->data)
    root->left = insert(root->left, data);
  else
    root->right = insert(root->right, data);
  return root;
}
                    (left->root->right) 15/2/24
void inorder(struct node *temp) {
  if (temp == NULL)
    return;
  inorder(temp->left);
  printf("%d", temp->data);
  inorder(temp->right);
}
```

(root→left→right

```
void preorder(struct node *temp){
  if(temp==NULL)
    return;  → printf("%d",temp→data);
  preorder(temp→left);
  preorder(temp→right);
}
                    (left→right→root)
void postorder(struct node *temp){
  if(temp==NULL)
  return;
  postorder(temp→left);
  postorder(temp→right);
  printf("%d",temp→data);
}
int main()
{
  struct node *root=NULL;
  int data,choice;
  do{
  printf("Enter your choice");
  printf("1. Insert");
  printf("2. Inorder traversal");
  printf("3. Preorder traversal");
  printf("4. Postorder traversal");
  printf("5. Exit");
  scanf("%d",&choice);
switch(choice)
{
  case 1: printf("Enter the value to be inserted.");
          scanf("%d",&data);
          root=insert(root,data);
          break;
case 2: printf("Inorder traversal of binary tree");
          inorder(root);
          break;
```

```
case 3: printf("Preorder traversal of binary tree is \n");
        preorder(root);
        break;
case 4: printf("Postorder traversal of binary tree \n");
        postorder(root);
        break;
case 5: printf("Exit");
        break;
default: printf("Invalid choice. Please try again");
}
} while(choice!=5)
return 0;
}
```

output

main menu
1. insert
2. inorder traversal
3. postorder traversal
4. preorder traversal
5. Exit

Enter the choice: 1
enter the value to be inserted: 100

Enter the choice: 1
Enter the value to be inserted: 20

Enter the choice: 1
Enter the value to be inserted 200

Enter the choice: 1
Enter the value to be inserted: 10

Enter the choice: 1
Enter the value to be inserted: 30

Enter the choice: 1
Enter the value to be inserted: 150

Enter the choice: 1
Enter the value to be inserted: 300

Enter the choice: 2
Inorder traversal: 10 20 30 100 150 200 300

Enter the choice: 3
Postorder traversal: 10 30 20 150 300 200 100

Enter the choice: 4
Preorder traversal: 100 20 10 30 200 150 300

Split linked list in parts

```
typedef struct listNode lnode;
int get_len (lnode * head)
{
  int n=0;
  while(head)
  {
    n++;
    head = head->next;
  }
  return;
}
struct listNode ** splitListToParts(struct listNode
*head, int k, int * returnSize)
{
  int n = get_len(head), elems, i, j;
  *returnSize = k;
  lnode **list = (lnode **) calloc (k, sizeof(lnode*));
    *t = head.
  if(n > k)
  {
    for(i=0; i<k; i++)
    {
      elems = i<n%k ? n/k+1 : n/k;
      j=0.
      list[i] = head;
      t = head.
      while(j++i<elems)
      {
        t = head.
        head = head->next;
      }
      t->next = NULL;
    }
```

15/2/24.

```
core
{
  for(i=0; i<n; i++)
  {
    list[i]=head;
    head=head->next;
    list[i]->next=NULL;
  }
}
return list;
}
```

O/P

I/P: head=[1,2,3];
K=5

Output: ([1],[2],[3],[],[])

I/P head=[1,2,3,4,5,6,7,8,9,10]
K=3

Output=([1,2,3,4],[5,6,7],[8,9,10])

22/02/24 Week

a) Write BFS

#incl
#defi

int n
queue
void
vi
qu
w

fo

}
}
}
}
int
e
f
f
f
}
fo
f

22/02/24 Week 9

a) Write a program to traverse a graph using BFS

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

int n, i, j, visited[MAX_VERTICES],
queue[MAX_VERTICES], front=0, rear=0;
void bfs(int v){
  visited[v]=1;
  queue[rear++]=v;
  while(front<rear){
    int current = queue[front++];
    printf("%d\t", current);
    int i;
   for(i=0; i<n; i++){
    if(adj[current][i] && !visited[i]){
      visited[i]=1;
      queue[rear++]=i;
    }
   }
  }
}
int main(){
  int v;
  printf("Enter the number of vertices");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  visited[i]=0;
}
printf("Enter the graph data in matrix form");
for(i=0; i<n; i++)
  for(j=0; j<n; j++)
    scanf("%d", &adj[i][j]);
```

```
printf("Enter the starting vertex: ");
scanf("%d", &v);
dfs(v);
for(i=0; i<n; i++){
  if(!visited[i])
    printf("\nBFS is not possible. Not all
            nodes are reachable\n");
  return 0;
}
}
return 0;
}
```

Output:

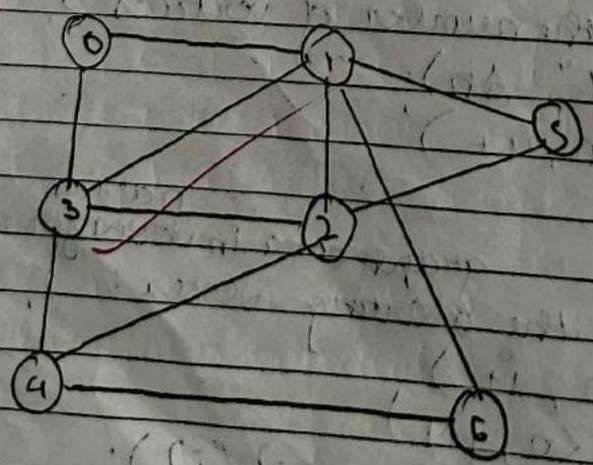
Enter the number of vertices: 7

Enter the adjacency matrix:

```
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
```

Enter the starting vertex: 2

2 1 3 4 5 0 6



(5) Write a program to check whether the given graph is connected or not using DFS method

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 10

void dfs (int graph[MAX_VERTICES][MAX_VERTICES],
int num_vertices, bool visited[MAX_vertices],
int vertex) {
    visited[vertex] = true;
    int i;
    for (i=0; i< numVertices ; ++i ) {
     if (graph[vertex[i]] == 1 && !visited[i]) {
      dfs (graph, num_vertices, visited, i );
    }
  }
}
bool_is_connected (int graph[MAX_VERTICES]
[MAX_VERTICES], int num_vertices ) {
   bool visited [MAX_VERTICES] = {false};
   dfs (graph, num_vertices, visited, 0);
   int i;
   for (i=0; i< num_vertices; ++i) {
     if (!visited[i]) {
        return false;
     }
   }
   return true;
}
```

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

Output:

Enter the number of vertices: 7

Enter the adjacency matrix:

0 1 0 1 0 0 0

1 0 1 1 0 1 1

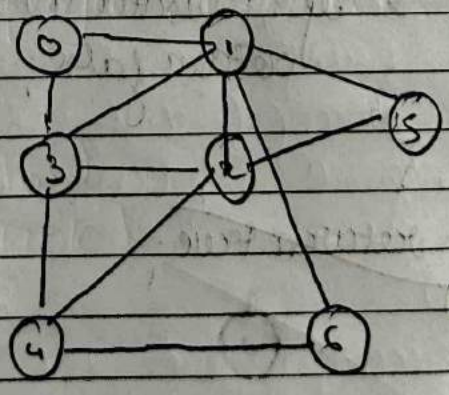
0 1 0 1 1 1 0

1 1 1 0 1 0 0

0 0 1 1 0 0 1

0 1 1 0 0 0 0

0 1 0 0 1 0 0



The graph is connected

Rotate list

```
int GetLength(struct distNode * head)
{
  if (head == NULL)
    return 0;
  return 1 + GetLength(head->next);
}
struct distNode * rotateRight(struct distNode *head, int k)
{
  if (head == NULL || k == 0)
    return head;
  int length = GetLength(head);
  if (length == 1)
    return head;
  for (int i = 0; i < k % length; i++)
  {
    struct distNode * p = head;
    while (p->next->next != NULL)
    {
      p = p->next;
    }
    struct distNode * a = (struct distNode *) malloc
    (sizeof (struct distNode));
    a->val = p->next->val;
    a->next = head;
    head = a;
    p->next = NULL;
  }
  return head;
}
```

Output

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

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

head = [10, 1, 2] k = 4

29/02/24 Week-10 : Hashing

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_EMPLOYEES 100
#define HASH_TABLE_SIZE 10
struct Employee {
    int key;
};

int hashFunction(int key);
void insertEmployee(struct Employee employees[],
int hashTable[], struct Employee emp);
void displayHashTable(int hashTable[]);

int main(){
    struct Employee employees[MAX_EMPLOYEES];
    int hashTable[HASH_TABLE_SIZE]={0};
    int n, m, i;
    printf("Enter the number of employees: ");
    scanf("%d", &n);
    printf("Enter employee records:\n");
    for(i=0; i<n; i++){
        printf("Employee %d:\n", i+1);
        printf("Enter 4-digit key: ");
        scanf("%d", &employees[i].key);
        insertEmployee(employees, hashTable,
        employees[i]);
    }
    printf("\nHash table:\n");
    displayHashTable(hashTable);
    return 0;
}

int hashFunction(int key){
    return key % HASH_TABLE_SIZE;
}
```

```
void insertEmployee (struct Employee employees[],
int hashTable[], struct Employee emp) {
  int index = hashFunction (emp.key);
  while (hashTable[index] != 0) {
    index = (index + 1) % HASH_TABLE_SIZE;
  }
  hashTable[index] = emp.key;
}
void displayHashTable (int hashTable[]) {
  int i;
  for (i=0; i<HASH_TABLE_SIZE; ++i) {
    printf ("%d -> ", i);
    if (hashTable[i] == 0) {
      printf ("Empty\n");
    } else {
      printf ("%d\n", hashTable[i]);
    }
  }
}
```

Output:

```
Enter the number of employees: 10
Enter the employee records:
Employee 1:
Enter 4-digit key 1234
Employee 2
Enter 4-digit key: 3456
Employee 3
Enter 4-digit key: 2675
Employee 4
Enter 4-digit key: 2789
Employee 5
Enter 4 digit key: 1254
```

Employee 6
Enter 4-digit key : 8970
Employee 7
Enter 4-digit key : 4678
Employee 8
Enter 4-digit key : 3241
Employee 9
Enter 4-digit key : 9875
Employee 10
Enter 4-digit key : 2456

Hash table
0 → 8970
1 → 3241
2 → 9875
3 → 2456
4 → 1234
5 → 2675
6 → 3456
7 → 1254
8 → 4578
9 → 2789

Sakshi 29/2/24

Hackerrank

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int id;
  int depth;
  struct node *left, *right;
};
void inorder(struct node *tree)
{
  if (tree == NULL)
    return;
  inorder(tree->left);
  printf("%d", tree->id);
  inorder((tree->right));
}
int main(void)
{
  int no_of_nodes, i=0;
  int l, r, max_depth, k;
  struct node *temp=NULL;
  scanf("%d", &no_of_nodes);
  struct node *tree = (struct node *) calloc
  (no_of_nodes, sizeof(struct node));
  tree[0].depth = 1;
  while (i < no_of_nodes)
  {
    tree[i].id = i+1;
    scanf("%d %d", &l, &r);
    if (l == -1)
      tree[i].left = NULL;
  else
  {
    tree[i].left = &tree[l-1];
```

```
tree[i].right->depth = tree[i].depth+1;
max_depth = tree[i].right->depth+2;
}
i++;
}
scanf("%d",&i);
while(i--)
{
scanf("%d",&k);
x=k;
while(k<=max_depth)
{
for(k=0; k<no_of_nodes;++k)
{
if(tree[k].depth==k)
{
temp=tree[k].left;
tree[k].left=tree[k].right;
tree[k].right=temp;
}
}
k=k+x;
}
inorder(tree);
printf("\n");
}
return 0;
}
```

OUTPUT :

Input
3
2 8
-1 -1
-1 -1
2
1
1

Output
3 1 2
2 1 3

Input
17
2 8
4 5
6 -1
-1 7
8 9
10 11
12 13
-1 14
-1 -1
15 -1
16 12
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
2

Output
14 8 5 9 2 4 13 7 12 1 3
10 16 6 17 11 15 9 5 14 8 2
13 7 12 4 1 8 17 11 16 6 [0]15

29/2/24