VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Navya Billalar (1BM22CS175)

in partial fulfilment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Dec 2023- March 2024

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out Navya Billalar **(1BM22CS175)**, who is a bonafide student of **B. M. S. College of Engineering**. It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST) work** prescribed for the said degree.

Prof. Sneha S Bagalkot
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyse data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Week 0 (07-12-2023):

Q1)

Develop a C program that simulates a basic banking system with functionalities like account creation, withdrawal, deposit, and balance inquiry. Write different user-defined function for each.

Ans)

```
#include<stdio.h>
void main()
{
   int option, ids[100], acnos[100], pins[100], bals[100];
   int c_ids=0, c_acnos=0, c_pins=0, c_bals=0,t=0;
   while (t==0)
   {
      printf("\nEnter 1 to create an account\n");
      printf("Enter 2 to withdraw an amount\n");
      printf("Enter 3 to deposit an amount\n");
      printf("Enter 4 for balance inquiry\n");
      printf("Enter 0 to exit\n");
      printf("\nEnter an option:\n");
      scanf("%d",&option);
      switch (option)
      {
         case 0: printf("\nExiting...\n");t=1;break;
         case 1:
create(ids,bals,pins,acnos,c_ids,c_bals,c_pins,c_acnos);c_ids+=1;c_bals+=1;c_pins+=1;c_ac
nos+=1;break;
         case 2: withdraw(acnos,pins,bals,c_acnos);break;
         case 3: deposit(acnos,bals,c_acnos);break;
         case 4: inquiry(acnos,bals,c_acnos);break;
      }
   }
}
void create(int Id[],int Amt[],int Pin[],int Acno[],int c1,int c2,int c3,int c4)
```

```
{
   int id_no,amt1,pin1;
   printf("Enter the id number of the applicant:\n");
   scanf("%d",&id_no);
   Id[c1]=id_no;
   Acno[c4]=id_no+2023;
   printf("Enter the initial amount to deposit:\n");
   scanf("%d",&amt1);
   Amt[c2]=amt1;
   printf("Enter a pin number:\n");
   scanf("%d",&pin1);
   Pin[c3]=pin1;
   printf("\nAccount details recorded successfully\n");
   printf("Your Account number is %d\n\n",Acno[c4]);
}
void withdraw(int Acno[],int Pin1[],int Amt[],int c1)
{
   int amount,pin,acc_no,i,flag=0;
   printf("Enter the account number:\n");
   scanf("%d",&acc_no);
   printf("Enter the amount to withdraw:\n");
   scanf("%d",&amount);
   printf("Enter your pin number:\n");
   scanf("%d",&pin);
   for(i=0;i<=c1;i++)
   {
      if (Acno[i]==acc_no && Pin1[i]==pin)
      {
         Amt[i]=Amt[i]-amount;
         printf("Please collect your cash\n");
         flag=1;
         break;
```

```
        }
        else
            continue;
    }
    if (flag==0)
        printf("Account does not exist or wrong details entered\n");
}
void deposit(int Acno[],int Amt[],int c1)
{
    int amount,acc_no,i,flag=0;
    printf("Enter the account number:\n");
    scanf("%d",&acc_no);
    printf("Enter the amount to deposit:\n");
    scanf("%d",&amount);
    for(i=0;i<=c1;i++)
    {
        if (Acno[i]==acc_no)
        {
            Amt[i]=Amt[i]+amount;
            printf("Deposited successfully\n");
            flag=1;
            break;
        }
        else
            continue;
    }
    if (flag==0)
        printf("Account does not exist or wrong details entered\n");
}
void inquiry(int Acno[],int Amt[],int c1)
{
    int acc_no,balance,i,flag=0;
```

```
    printf("Enter the account number:\n");
    scanf("%d",&acc_no);
    for(i=0;i<=c1;i++)
    {
       if (Acno[i]==acc_no)
       {
          balance=Amt[i];
          printf("The balance is %d\n",balance);
          flag=1;
          break;
       }
       else
          continue;
    }
    if (flag==0)
       printf("Account does not exist or wrong details entered\n");
}
```

Output:

```
Enter 1 to create an account
Enter 2 to withdraw an amount
Enter 3 to deposit an amount
Enter 4 for balance inquiry
Enter 0 to exit

Enter an option:
1
Enter the id number of the applicant:
1234
Enter the initial amount to deposit:
5000
Enter a pin number:
1111

Account details recorded successfully
Your Account number is 3257
```

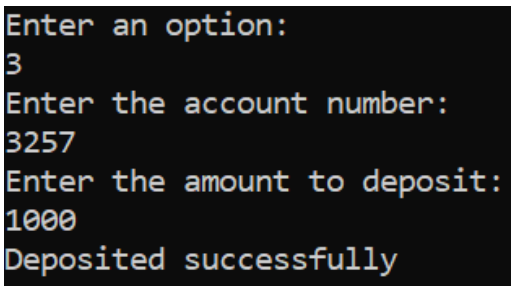
```
Enter 1 to create an account
Enter 2 to withdraw an amount
Enter 3 to deposit an amount
Enter 4 for balance inquiry
Enter 0 to exit

Enter an option:
2
Enter the account number:
3257
Enter the amount to withdraw:
2000
Enter your pin number:
1111
Please collect your cash
```

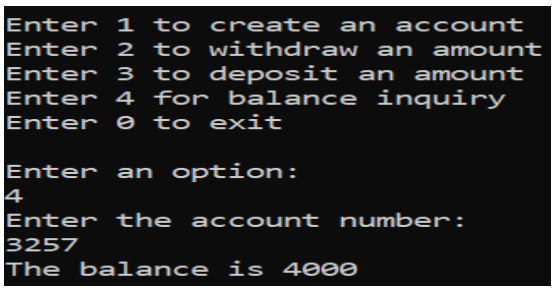
Q2)

Implement a C program that sorts strings lexicographically, considering uppercase and lowercase letters, and without using the standard library sorting functions.

Ans)

```
#include<stdio.h>
#include<string.h>
void main()
{
```






```
   int n,i;
   printf("Enter the number of strings:\n");
   scanf("%d",&n);
   char st[n][100];
   printf("Enter the strings:\n");
   for(i=0;i<n;i++)
   {
      scanf("%s",st[i]);
   }
   sort_string(st,n);
}
void sort_string(char arr[][100],int n)
{
   char temp[50];
   int i,j;
   for(i=0;i<n-1;i++)
   {
      for(j=i+1;j<n;j++)
```

```
        {
            if (strcmp(arr[i],arr[j])>0)
            {
                strcpy(temp,arr[i]);
                strcpy(arr[i],arr[j]);
                strcpy(arr[j],temp);
            }
        }
    }
    printf("The lexicographically sorted strings are:\n");
    for(i=0;i<n;i++)
        printf("%s\n",arr[i]);
}
```

Output:

```
Enter the number of strings:
5
Enter the strings:
ABC
def
Ghi
jKL
mNo
The lexicographically sorted strings are:
ABC
Ghi
def
jKL
mNo
```

Q3)

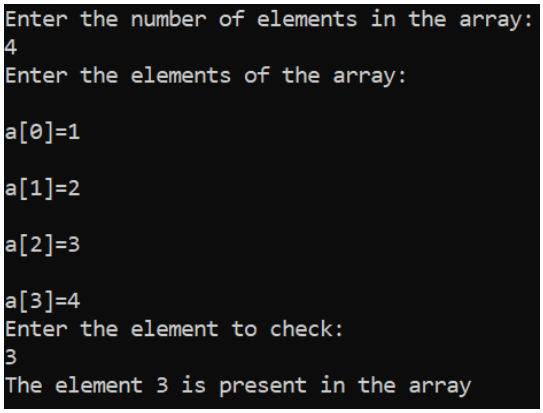
Implement a C program to check if a given element is present in a 2D array with a user defined function.

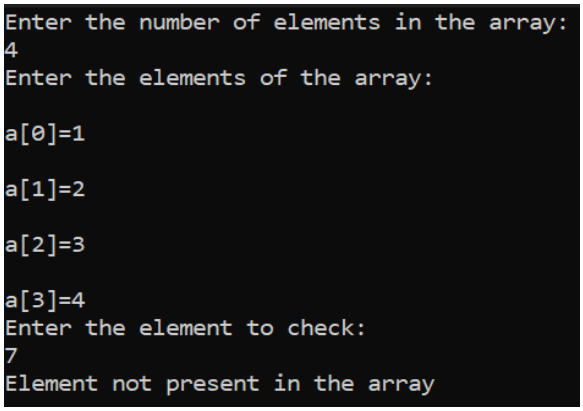
Ans)

```
#include<stdio.h>
void main()
{
    int a[100],n,i,number;
    printf("Enter the number of elements in the array:\n");
```

```
    scanf("%d",&n);
    printf("Enter the elements of the array:\n");
    for(i=0;i<n;i++)
    {
        printf("\na[%d]=",i);
        scanf("%d",&a[i]);
    }
    printf("Enter the element to check:\n");
    scanf("%d",&number);
    check(a,number,n);
}
void check(int arr[],int num,int num1)
{
    int i,flag=0;
    for(i=0;i<num1;i++)
    {
        if (arr[i]==num)
        {
            printf("The element %d is present in the array\n",num);
            flag=1;
        }
    }
    if (flag==0)
        printf("Element not present in the array\n");
}
```

Output:





Q4)

Create a program in C to search for a substring within a larger string with a user defined function.

Ans)

```
#include<stdio.h>
#include<string.h>
void main()
{
   char str1[100],str2[100];
   printf("Enter the larger string:\n");
   gets(str1);
   printf("Enter the substring:\n");
   gets(str2);
   check(str1,str2);
}
void check(char s1[],char s2[])
{
   int i=0,j=0;
   while (s1[i]!='\0' && s2[j]!='\0')
   {
      if (s1[i]!=s2[j])
      {
```

```
            i++;
            j=0;
        }
        else
        {
            j++;
            i++;
        }
    }
    if (s2[j]=='\0')
        printf("The substring is found in the larger string\n");
    else
        printf("The substring is not found in the larger string\n");
}
```

Output:

```
Enter the larger string:
BMSCollegeOfEngineering
Enter the substring:
OfE
The substring is found in the larger string
```

```
Enter the larger string:
BMSCollegeOfEngineering
Enter the substring:
abc
The substring is not found in the larger string
```

Q5)

Write a C program to find the index of the last occurrence of a number in an array with a user defined function.

Ans)

```
#include<stdio.h>
void main()
{
    int a[100],i,n,num1;
    printf("Enter the number of elements in the array:\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
```

```
    {
        printf("\na[%d]=",i);
        scanf("%d",&a[i]);
    }
    printf("Enter the number whose index of last occurrence has to be found:\n");
    scanf("%d",&num1);
    index1(a,n,num1);
}
void index1(int arr[],int n1,int num2)
{
    int i,index,flag=0;
    for(i=0;i<n1;i++)
    {
        if (arr[i]==num2)
        {
            index=i;
            flag=1;
        }
    }
    if (flag==0)
        printf("The element does not occur even once in the array\n");
    else
        printf("The index of the last occurrence of the number is %d\n",index);
}
```

Output:

```
Enter the number of elements in the array:
7

a[0]=1

a[1]=2

a[2]=3

a[3]=4

a[4]=1

a[5]=3

a[6]=4
Enter the number whose index of last occurrence has to be found:
3
The index of the last occurrence of the number is 5
```

Q6)

Write a C program to search for a specific element in an array using linear search with a user defined function.

Ans)

```
#include<stdio.h>
void main()
{
   int a[100],i,n,num1;
   printf("Enter the number of elements in the array:\n");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("\na[%d]=",i);
      scanf("%d",&a[i]);
   }
   printf("Enter the number to find:\n");
   scanf("%d",&num1);
   linear_search(a,n,num1);
}
```

```
void linear_search(int arr[],int n1,int num2)
{
   int i,pos,flag=0;
   for(i=0;i<n1;i++)
   {
      if (arr[i]==num2)
      {
         pos=i+1;
         printf("The number is found at position %d\n",pos);
         flag=1;
      }
   }
   if (flag==0)
      printf("The number is not present in the array\n");
}
```

Output:

```
Enter the number of elements in the array:
5

a[0]=10

a[1]=20

a[2]=30

a[3]=40

a[4]=50
Enter the number to find:
40
The number is found at position 4
```

Q7)

Implement a C program to perform a binary search on a sorted array with a user defined function.

Ans)

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

```
void main()
{
   int a[100],i,n,num1;
   printf("Enter the number of elements in the array:\n");
   scanf("%d",&n);
   printf("Enter the elements in a sorted manner\n");
   for(i=0;i<n;i++)
   {
      printf("\na[%d]=",i);
      scanf("%d",&a[i]);
   }
   printf("Enter the number to find:\n");
   scanf("%d",&num1);
   binary_search(a,n,num1);
}
void binary_search(int arr[],int n1, int num2)
{
   int beg=0,end=n1-1,mid,pos,flag=0;
   while (beg<=end)
   {
      mid=(beg+end)/2;
      if (arr[mid]==num2)
      {
         pos=mid+1;
         printf("The number is found at position %d\n",pos);
         flag=1;
         break;
      }
      if (arr[mid]>num2)
         end=mid-1;
```

```
    else
        beg=mid+1;
  }
  if (flag==0)
    printf("The element is not found in the array\n");
}
```

Output:

```
Enter the number of elements in the array:
5
Enter the elements in a sorted manner

a[0]=12

a[1]=34

a[2]=45

a[3]=67

a[4]=78
Enter the number to find:
45
The number is found at position 3
```

Q8)

Create a program in C to search for the minimum and maximum elements in an array with a user defined function.

Ans)

```
#include<stdio.h>
void main()
{
  int a[100],i,n;
  printf("Enter the number of elements in the array:\n");
  scanf("%d",&n);
  printf("Enter the elements of the array:\n");
  for(i=0;i<n;i++)
  {
```

```
        printf("\na[%d]=",i);
        scanf("%d",&a[i]);
    }
    printf("\nThe minimum element in the array is:\n");
    min_ar(a,n);
    printf("\nThe maximum element in the array is:\n");
    max_ar(a,n);
}
void min_ar(int arr[],int num)
{
    int m=999999;
    int i;
    for(i=0;i<num;i++)
    {
        if (arr[i]<m)
            m=arr[i];
    }
    printf("%d",m);
}
void max_ar(int arr[],int num)
{
    int m=-999999;
    int i;
    for(i=0;i<num;i++)
    {
        if (arr[i]>m)
            m=arr[i];
    }
    printf("%d",m);
}
```

Output:

```
Enter the number of elements in the array:
5
Enter the elements of the array:

a[0]=11

a[1]=26

a[2]=47

a[3]=69

a[4]=82

The minimum element in the array is:
11
The maximum element in the array is:
82
```

Week 1 (21-12-2023):

Q1)

Write a C program to swap two numbers using pointers.

Ans)

```
#include <stdio.h>
void swapNumbers(int *a,int *b)
{
   int temp=*a;
   *a=*b;
   *b=temp;
}
int main()
{
   int n1,n2;
   printf("Enter the first number: ");
   scanf("%d", &n1);
   printf("Enter the second number: ");
```

```
    scanf("%d", &n2);
    printf("Before swapping: \n");
    printf("First number: %d\n",n1);
    printf("Second number: %d\n",n2);
    swapNumbers(&n1, &n2);
    printf("\nAfter swapping: \n");
    printf("First number: %d\n", n1);
    printf("Second number: %d\n", n2);
    return 0;
}
```

Output:

```
Enter the first number: 2
Enter the second number: 3
Before swapping:
First number: 2
Second number: 3

After swapping:
First number: 3
Second number: 2
```

Q2)

Write a C program to demonstrate dynamic memory allocation.

Ans)

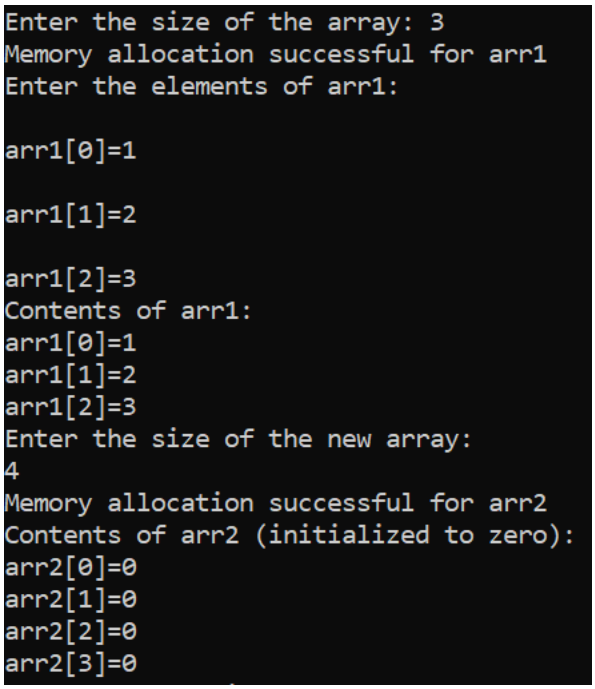
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *arr1,*arr2;
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    arr1=(int *)malloc(n*sizeof(int));
```

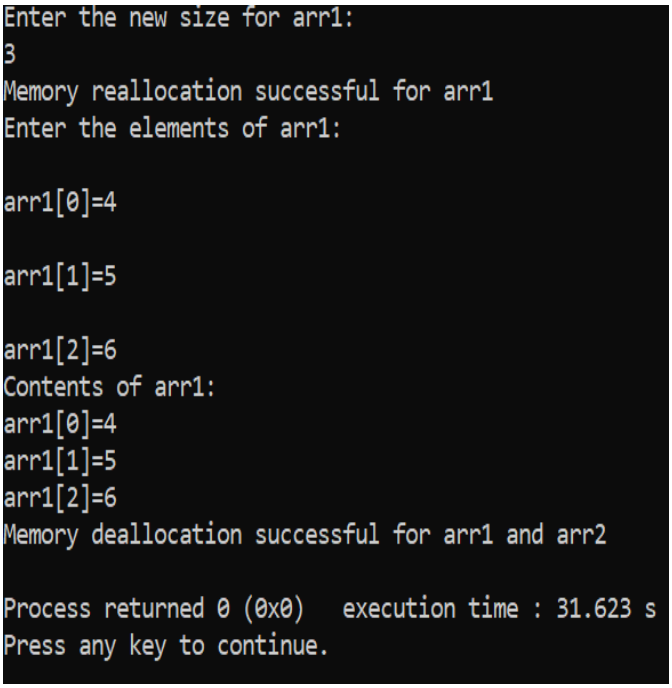
```
if(arr1==NULL)
{
   printf("Memory allocation failed for arr1\n");
   return 1;
}
printf("Memory allocation successful for arr1\n");
printf("Enter the elements of arr1:\n");
for(int i=0;i<n;i++)
{
   printf("\narr1[%d]=",i);
   scanf("%d", &arr1[i]);
}
printf("Contents of arr1: ");
for(int i=0;i<n;i++)
{
   printf("\narr1[%d]=%d",i,arr1[i]);
}
printf("\n");
//free(arr1);
//printf("Memory deallocation successful for arr1\n");
printf("Enter the size of the new array:\n");
scanf("%d", &n);
arr2=(int *)calloc(n,sizeof(int));
if(arr2==NULL)
{
   printf("Memory allocation failed for arr2\n");
   return 1;
}
printf("Memory allocation successful for arr2\n");
printf("Contents of arr2 (initialized to zero): ");
```

```
for(int i=0;i<n;i++)
{
   printf("\narr2[%d]=%d ",i,arr2[i]);
}
printf("\n");
printf("Enter the new size for arr1:\n");
scanf("%d",&n);
arr1 = (int *)realloc(arr1,n*sizeof(int));
if(arr1==NULL)
{
   printf("Memory reallocation failed for arr1\n");
   return 1;
}
printf("Memory reallocation successful for arr1\n");
printf("Enter the elements of arr1:\n");
for(int i=0;i<n;i++)
{
   printf("\narr1[%d]=",i);
   scanf("%d",&arr1[i]);
}
printf("Contents of arr1: ");
for(int i=0;i<n;i++)
{
   printf("\narr1[%d]=%d",i,arr1[i]);
}
printf("\n");
free(arr1);
free(arr2);
printf("Memory deallocation successful for arr1 and arr2\n");
return 0;
```

}

Output:





Q3)

Write a C program to demonstrate stack implementation.

Ans)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 5
struct Stack
{
   int arr[MAX_SIZE];
   int top;
};
void initialize(struct Stack *stack);
void push(struct Stack *stack);
void pop(struct Stack *stack);
void display(const struct Stack *stack);
```

```
int main()
{
   struct Stack stack;
   int c,c1=0;
   initialize(&stack);
   while(c1==0)
   {
      printf("Enter 1 to push an element into the stack\n");
      printf("Enter 2 to pop an element from the stack\n");
      printf("Enter 3 display the contents of the stack\n");
      printf("Enter 0 to exit\n");
      printf("Enter your choice");
      scanf("%d",&c);
      switch(c)
      {
         case 0: c1=1;break;
         case 1: push(&stack);break;
         case 2: pop(&stack);break;
         case 3: display(&stack);break;
         default: printf("Invalid choice\n");break;
      }
   }
   return 0;
}
void initialize(struct Stack *stack)
{
   stack->top=-1;
}
void push(struct Stack *stack)
{
```

```
    int value;
    printf("Enter the element to push into the stack:\n");
    scanf("%d",&value);
    if(stack->top==MAX_SIZE-1)
    {
        printf("Stack Overflow: Cannot push %d, stack is full.\n",value);
    }
    else
    {
        stack->arr[++(stack->top)]=value;
        printf("Pushed %d onto the stack.\n",value);
    }
}
void pop(struct Stack *stack)
{
    int poppedElement;
    if (stack->top==-1)
    {
        printf("Stack Underflow: Cannot pop from an empty stack.\n");
    }
    else
    {
        poppedElement=stack->arr[(stack->top)--];
        printf("The popped element is: %d\n",poppedElement);
    }
}
void display(const struct Stack *stack)
{
    if (stack->top==-1)
    {
```

```
        printf("Stack is empty.\n");
    }
    else
    {
        printf("Stack Contents: ");
        for (int i=0;i<=stack->top;i++)
        {
            printf("%d ",stack->arr[i]);
        }
        printf("\n");
    }
}
```

Output:

```
Enter 1 to push an element into the stack
Enter 2 to pop an element from the stack
Enter 3 display the contents of the stack
Enter 0 to exit
Enter your choice1
Enter the element to push into the stack:
1
Pushed 1 onto the stack.
Enter 1 to push an element into the stack
Enter 2 to pop an element from the stack
Enter 3 display the contents of the stack
Enter 0 to exit
Enter your choice1
Enter the element to push into the stack:
2
Pushed 2 onto the stack.
Enter 1 to push an element into the stack
Enter 2 to pop an element from the stack
Enter 3 display the contents of the stack
Enter 0 to exit
```

```
Enter your choice3
Stack Contents: 1 2
Enter 1 to push an element into the stack
Enter 2 to pop an element from the stack
Enter 3 display the contents of the stack
Enter 0 to exit
Enter your choice2
The popped element is: 2
Enter 1 to push an element into the stack
Enter 2 to pop an element from the stack
Enter 3 display the contents of the stack
Enter 0 to exit
Enter your choice3
Stack Contents: 1
Enter 1 to push an element into the stack
Enter 2 to pop an element from the stack
Enter 3 display the contents of the stack
Enter 0 to exit
Enter your choice
```

Week 2 (28-12-2023):

Q1)

Write a C program to convert an infix expression to a postfix expression.

Ans)

#include<stdio.h>

```
#include<stdlib.h>
#define Max 100
char stack[Max];
int top=-1;
void push(char item)
{
   if(top==Max-1)
   {
      printf("Stack Overflow\n");
      exit(EXIT_FAILURE);
   }
   stack[++top]=item;
}
char pop()
{
   if(top==-1)
   {
      printf("Stack Underflow\n");
      exit(EXIT_FAILURE);
   }
   return stack[top--];
}
int precedence(char operator)
{
   switch(operator)
   {
      case '^': return 3;
      case '*':
      case '/':
         return 2;
```

```
        case '+':
        case '-':
            return 1;
        default: return 0;
    }
}
int main()
{
    char infix[Max];
    printf("Enter a valid parenthesized infix expression: ");
    scanf("%s",infix);
    char postfix[Max];
    int i,j;
    i=j=0;
    while(infix[i]!='\0')
    {
        char symbol=infix[i];
        if((symbol>='a' && symbol<='z')||(symbol>='A' && symbol<='Z'))
            postfix[j++]=symbol;
        else if(symbol=='(')
            push(symbol);
        else if(symbol==')')
        {
            while(top!=-1 && stack[top]!='(')
                postfix[j++]=pop();
            if(top==-1)
            {
                printf("Invalid Expression due to parentheses mismatch\n");
                exit(EXIT_FAILURE);
            }
```

```
            pop();
        }
        else
        {
            while(top!=-1 && precedence(stack[top])>=precedence(symbol))
                postfix[j++]=pop();
            push(symbol);
        }
        i++;
    }
    while(top!=-1)
    {
        if(stack[top]=='(')
        {
            printf("Invalid Expression due to parentheses mismatch\n");
            exit(EXIT_FAILURE);
        }
        postfix[j++]=pop();
    }
    postfix[j]='\0';
    printf("Postfix Expression: %s\n", postfix);
    return 0;
}
```

Output:

```
Enter a valid parenthesized infix expression: A*B+C*D-E
Postfix Expression: AB*CD*+E-
```

Q2)

Write a C program to evaluate a postfix expression.

Ans)

```
#include <stdio.h>
#include <stdlib.h>
#define Max 100
int stack[Max];
int top=-1;
void push(int element)
{
   if(top==Max-1)
   {
      printf("Stack Overflow\n");
      exit(EXIT_FAILURE);
   }
   stack[++top]=element;
}
int pop()
{
   if(top==-1)
   {
      printf("Stack Underflow\n");
      exit(EXIT_FAILURE);
   }
   return stack[top--];
}
int main()
{
   char postfix[100];
   printf("Enter a postfix expression: ");
   scanf("%s", postfix);
   for(int i=0;postfix[i]!='\0';i++)
   {
```

```
        if(isdigit(postfix[i]))
        {
           push(postfix[i]-'0');
        }
        else
        {
           int op2=pop();
           int op1=pop();
           switch(postfix[i])
           {
              case '+': push(op1 + op2);break;
              case '-': push(op1 - op2);break;
              case '*': push(op1 * op2);break;
              case '/': push(op1 / op2);break;
              default: printf("Invalid operator\n");exit(EXIT_FAILURE);
           }
        }
    }
    int result=pop();
    printf("Result: %d\n",result);
    return 0;
}
```

Output:

```
Enter a postfix expression: 12*34*+5-
Result: 9
```

Q3)

Write a C program to demonstrate Linear Queue implementation.

Ans)

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

```
#define SIZE 5
int front=-1,rear=-1;
int queue[SIZE];
int main()
{
   int c,elem,c1=0;
   while(c1==0)
   {
      printf("\nEnter 1 to insert into the queue\n");
      printf("Enter 2 to delete from the queue\n");
      printf("Enter 3 to display the contents of the queue\n");
      printf("Enter 0 to exit\n");
      printf("\nEnter your choice:\n");
      scanf("%d",&c);
      switch(c)
      {
         case 0: printf("Exiting...\n");c1=1;break;
         case 1:
            printf("Enter the element to be inserted:\n");
            scanf("%d",&elem);
            enqueue(elem);
            break;
         case 2: dequeue();break;
         case 3: display_q();break;
         default: printf("Invalid choice. Please enter a valid option\n");break;
      }
   }
   return 0;
}
void enqueue(int elem)
```

```
{
   if(rear==SIZE-1)
   {
      printf("Queue is full. Cannot insert\n");
      exit(0);
   }
   if(front==-1)
   {
      front=0;
   }
   queue[++rear]=elem;
   printf("Element %d inserted successfully\n",elem);
}
void dequeue()
{
   if(front==-1||front>rear)
   {
      printf("Queue is empty. Cannot delete\n");
      exit(0);
   }
   printf("The element deleted is %d\n",queue[front]);
   front++;
}
void display_q()
{
   if(front==-1)
   {
      printf("Queue is empty\n");
      exit(0);
   }
```

```
    printf("The elements of the queue are:\n");
    for(int i=front;i<=rear;i++)
    {
        printf("%d",queue[i]);
    }
}
```

Output:

```
Enter 1 to insert into the queue
Enter 2 to delete from the queue
Enter 3 to display the contents of the queue
Enter 0 to exit

Enter your choice:
1
Enter the element to be inserted:
1
Element 1 inserted successfully

Enter 1 to insert into the queue
Enter 2 to delete from the queue
Enter 3 to display the contents of the queue
Enter 0 to exit

Enter your choice:
1
Enter the element to be inserted:
2
Element 2 inserted successfully

Enter 1 to insert into the queue
Enter 2 to delete from the queue
Enter 3 to display the contents of the queue
Enter 0 to exit
```

```
Enter your choice:
3
The elements of the queue are:
12
Enter 1 to insert into the queue
Enter 2 to delete from the queue
Enter 3 to display the contents of the queue
Enter 0 to exit

Enter your choice:
2
The element deleted is 1

Enter 1 to insert into the queue
Enter 2 to delete from the queue
Enter 3 to display the contents of the queue
Enter 0 to exit

Enter your choice:
3
The elements of the queue are:
2
```

Week 3 (11-01-2024):

Q1)

Write a C program to demonstrate Circular Queue implementation.

Ans)

```
#include<stdio.h>
#define SIZE 5
int front=-1,rear=-1;
```

```
int CQ[SIZE];
int main()
{
   int c,c1=0;
   while(c1==0)
   {
      printf("\nEnter 1 to insert into the circular queue\n");
      printf("Enter 2 to delete from the circular queue\n");
      printf("Enter 3 to display the contents of the queue\n");
      printf("Enter 0 to exit\n");
      printf("\nEnter your choice:\n");
      scanf("%d",&c);
      switch(c)
      {
         case 0: printf("Exiting...\n");c1=1;break;
         case 1: enqueue();break;
         case 2: dequeue();break;
         case 3: display_q();break;
         default: printf("Invalid choice. Please enter a valid option\n");
      }
   }
   return 0;
}
void enqueue()
{
   int elem;
   if(rear==SIZE-1)
   {
      printf("Queue is full. Cannot insert\n");
      exit(0);
```

```
    }
    if(front==-1)
    {
        front=0;
        rear=0;
    }
    printf("Enter the element to be inserted:\n");
    scanf("%d",&elem);
    CQ[rear]=elem;
    printf("Element %d inserted successfully\n",elem);
    rear=(rear+1)%SIZE;
}
void dequeue()
{
    if(front==-1)
    {
        printf("Queue is empty. Cannot delete\n");
        exit(0);
    }
    printf("The element deleted is %d\n",CQ[front]);
    front=(front+1)%SIZE;
}
void display_q()
{
    if(front==-1)
    {
        printf("Queue is empty\n");
        exit(0);
    }
    printf("The elements of the queue are:\n");
```

```
    for(int i=front;i<rear;i++)
    {
        printf("%d",CQ[i]);
    }
}
```

Output:

```
Enter 1 to insert into the circular queue
Enter 2 to delete from the circular queue
Enter 3 to display the contents of the queue
Enter 0 to exit

Enter your choice:
1
Enter the element to be inserted:
1
Element 1 inserted successfully

Enter 1 to insert into the circular queue
Enter 2 to delete from the circular queue
Enter 3 to display the contents of the queue
Enter 0 to exit

Enter your choice:
1
Enter the element to be inserted:
2
Element 2 inserted successfully
```

```
Enter 1 to insert into the circular queue
Enter 2 to delete from the circular queue
Enter 3 to display the contents of the queue
Enter 0 to exit

Enter your choice:
3
The elements of the queue are:
12
Enter 1 to insert into the circular queue
Enter 2 to delete from the circular queue
Enter 3 to display the contents of the queue
Enter 0 to exit

Enter your choice:
2
The element deleted is 1

Enter 1 to insert into the circular queue
Enter 2 to delete from the circular queue
Enter 3 to display the contents of the queue
Enter 0 to exit

Enter your choice:
3
The elements of the queue are:
2
```

Q2)

Write a C program to demonstrate Singly Linked List – Insert and Display implementation.

Ans)

```
#include<stdio.h>

#include<stdlib.h>

struct Node

{
```

```
    int data;
    struct Node *next;
};
int main()
{
    struct Node *head=NULL;
    int c=0,a,b;
    while(c!=3)
    {
        printf("\nEnter 1 to insert\n");
        printf("Enter 2 to display\n");
        printf("Enter 3 to exit\n");
        printf("\nEnter your choice:\n");
        scanf("%d",&c);
        switch(c)
        {
            case 1:
                printf("Enter the value to be inserted:\n");
                scanf("%d",&a);
                int c2;
                printf("\nEnter 1 to insert at the front\n");
                printf("Enter 2 to insert after a given node\n");
                printf("Enter 3 to insert at the end\n");
                printf("\nEnter your choice:\n");
                scanf("%d",&c2);
                switch(c2)
                {
                    case 1:
                        insertFront(&head,a);
                        break;
```

```
                case 3:
                    insertEnd(&head,a);
                    break;
                case 2:
                    printf("Enter the value after which to insert:\n");
                    scanf("%d",&b);
                    struct Node *temp=head;
                    while(temp!=NULL && temp->data!=b)
                        temp=temp->next;
                    if(temp==NULL)
                        printf("Element not found in the list\n");
                    else
                    {
                        insertMiddle(temp,a);
                    }
                    break;
                default: printf("Invalid Choice\n");break;
            }
        case 2: display(head);break;
        case 3: printf("Exiting...\n");break;
        default: printf("Invalid choice\n");break;

    }
  }
  return 0;
}
void insertFront(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);
    (*head)=new_node;
}
void insertMiddle(struct Node *previous,int new_data)
{
    if (previous==NULL)
    {
        printf("The previous node entered cannot be NULL\n");
        return;
    }
    struct Node *new_node=(struct Node *)malloc(sizeof(struct Node));
    new_node->data=new_data;
    new_node->next=previous->next;
    previous->next=new_node;
}
void insertEnd(struct Node **head,int new_data)
{
    struct Node *new_node=(struct Node *)malloc(sizeof(struct Node));
    struct Node *last=*head;
    new_node->data=new_data;
    new_node->next=NULL;
    if(*head==NULL)
    {
        *head=new_node;
        return;
    }
    while(last->next!=NULL)
        last=last->next;
    last->next=new_node;
}
```

```
void display(struct Node *node)
{
   printf("The contents of the list are:\n");
   while(node!=NULL)
   {
      printf("%d",node->data);
      node=node->next;
   }
   printf("\n");
}
```

Output:

```
Enter 1 to insert
Enter 2 to display
Enter 3 to exit

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

Enter 1 to insert at the front
Enter 2 to insert after a given node
Enter 3 to insert at the end

Enter your choice:
1
The contents of the list are:
1

Enter 1 to insert
Enter 2 to display
Enter 3 to exit

Enter your choice:
1
Enter the value to be inserted:
2

Enter 1 to insert at the front
Enter 2 to insert after a given node
Enter 3 to insert at the end

Enter your choice:
3
The contents of the list are:
12
```

```
Enter 1 to insert
Enter 2 to display
Enter 3 to exit

Enter your choice:
1
Enter the value to be inserted:
3

Enter 1 to insert at the front
Enter 2 to insert after a given node
Enter 3 to insert at the end

Enter your choice:
2
Enter the value after which to insert:
1
The contents of the list are:
132

Enter 1 to insert
Enter 2 to display
Enter 3 to exit

Enter your choice:
2
The contents of the list are:
132
```

Q3)

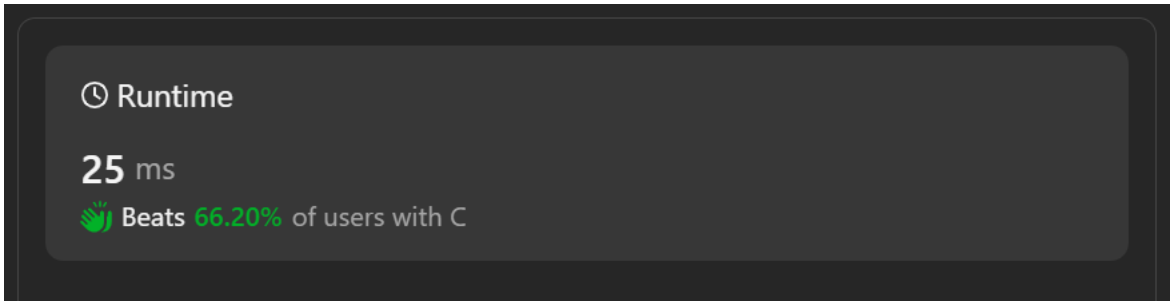
Leet Code on Singly Linked List

Ans)

```
typedef struct {
    int array[30000];
    int min[30000];
    int top1;
    int top2;
} MinStack;
MinStack* minStackCreate() {
    MinStack* obj=(MinStack*)malloc(sizeof(MinStack));
    obj->top1=-1;
    obj->top2=-1;
    return obj;
}
void minStackPush(MinStack* obj, int val) {
    obj->array[++obj->top1]=val;
    if(obj->top2==-1){
        obj->min[++obj->top2]=val;
        return;
    }
    int mintop=obj->min[obj->top2];
    if(mintop>val){
        obj->min[++obj->top2]=val;
        return;
    }
    else{
        obj->min[++obj->top2]=mintop;
    }
}
```

```
void minStackPop(MinStack* obj) {
    obj->top1--;
    obj->top2--;
}
int minStackTop(MinStack* obj) {
    return obj->array[obj->top1];
}
int minStackGetMin(MinStack* obj) {
    return obj->min[obj->top2];
}
void minStackFree(MinStack* obj) {
    free(obj);
}
```

Output:



Week 4 (18-01-2024):

Q1)

Write a C program to demonstrate Singly Linked List – Delete and Display implementation.

Ans)

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

struct Node
```

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

int main()
{
    struct Node* head = NULL;
    insertEnd(&head,1);
    insertEnd(&head,2);
    insertEnd(&head,3);
    insertEnd(&head,4);
    insertEnd(&head,5);
    insertEnd(&head,6);
    printf("Initial List:\n");
    display_q(head);
    int c,c1=0;
    while(c1==0)
    {
        printf("\nEnter 1 to delete from the beginning\n");
        printf("Enter 2 to delete at the end\n");
        printf("Enter 3 to delete from a specific position\n");
        printf("Enter 4 to display\n");
        printf("Enter 0 to exit\n");
        printf("\nEnter your choice:\n");
        scanf("%d",&c);
        switch(c)
        {
            case 0: printf("Exiting...\n");c1=1;break;
            case 1: delete_beg(&head);display_q(head);break;
```

```
            case 2: delete_end(&head);display_q(head);break;
            case 3: delete_mid(&head);display_q(head);break;
            case 4: display_q(head);break;
            default: printf("Invalid choice. Please enter a valid option\n");break;
        }
    }
    return 0;
}

void delete_beg(struct Node** head)
{
    if(*head==NULL)
    {
        printf("List is empty\n");
    }
    else
    {
        struct Node* ptr=*head;
        *head=(*head)->next;
        free(ptr);
        printf("Node deleted from beginning\n");
    }
}

void delete_end(struct Node** head)
{
    struct Node* ptr;
    struct Node* ptr1=NULL;
    if(*head==NULL)
    {
```

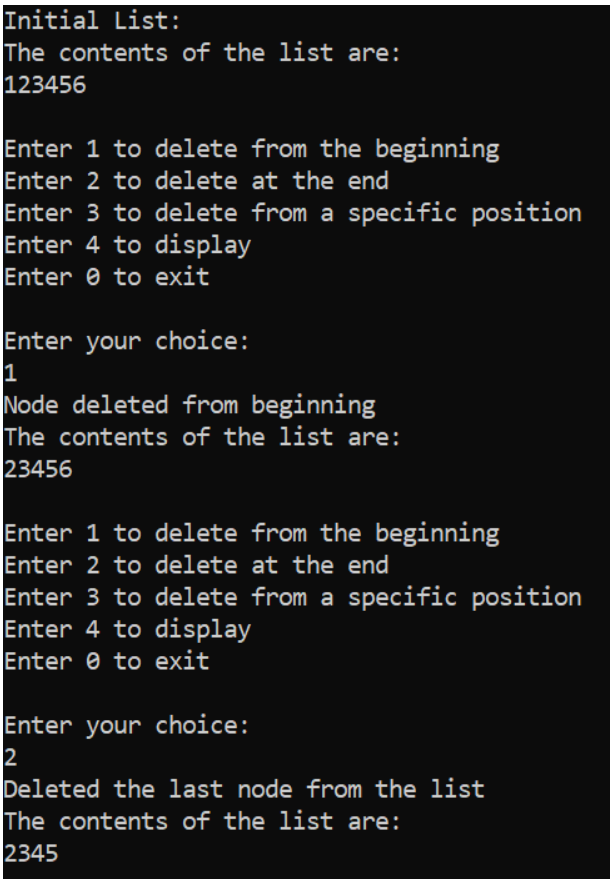
```
        printf("List is empty\n");
    }
    else if((*head)->next==NULL)
    {
        free(*head);
        *head=NULL;
        printf("Deleted the only node in the list\n");
    }
    else
    {
        ptr=*head;
        while(ptr->next!=NULL)
        {
            ptr1=ptr;
            ptr=ptr->next;
        }
        ptr1->next=NULL;
        free(ptr);
        printf("Deleted the last node from the list\n");
    }
}

void delete_mid(struct Node** head)
{
    struct Node* ptr;
    struct Node* ptr1=NULL;
    int loc;
    printf("Enter the location of the node to be deleted:\n");
    scanf("%d",&loc);
    ptr=*head;
```

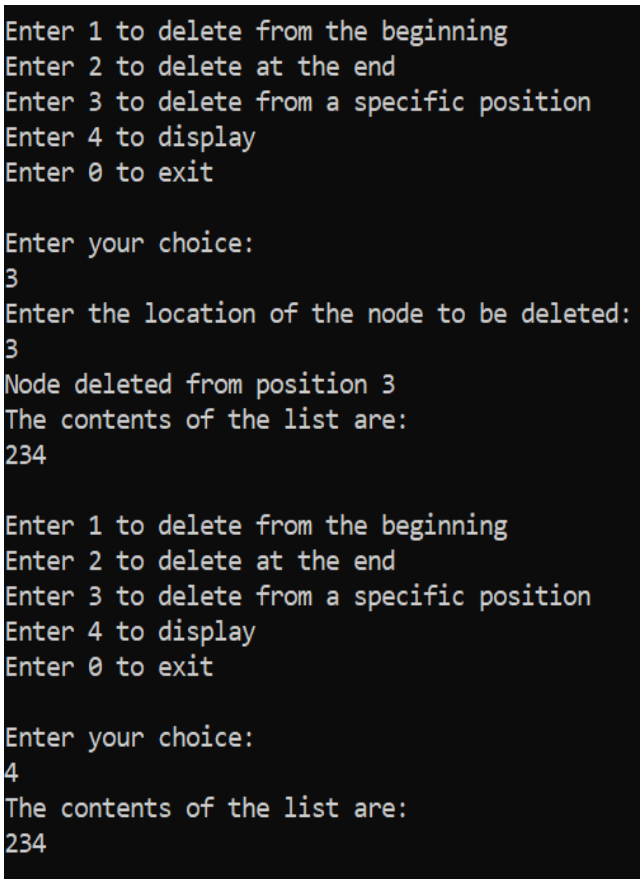
```
    for(int i=0;i<loc;i++)
    {
        ptr1=ptr;
        ptr=ptr->next;
        if(ptr==NULL)
        {
            printf("Less elements than required in the list\n");
            return;
        }
    }
    ptr1->next=ptr->next;
    free(ptr);
    printf("Node deleted from position %d\n",loc);
}

void display_q(struct Node* head)
{
    struct Node* current=head;
    if(current==NULL)
    {
        printf("The list is empty\n");
        return;
    }
    else
    {
        printf("The contents of the list are:\n");
        while(current!=NULL)
        {
            printf("%d",current->data);
            current=current->next;
```

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

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

void insertEnd(struct Node** head, int data)
{
    struct Node* newNode=createLinkedList(data);

    if(*head==NULL)
    {
        *head=newNode;
    }
    else
    {
        struct Node* current=*head;
        while(current->next != NULL)
        {
            current=current->next;
        }
        current->next=newNode;
    }
```

}

Output:





Q2)

Leet Code – Singly Linked List

Ans)

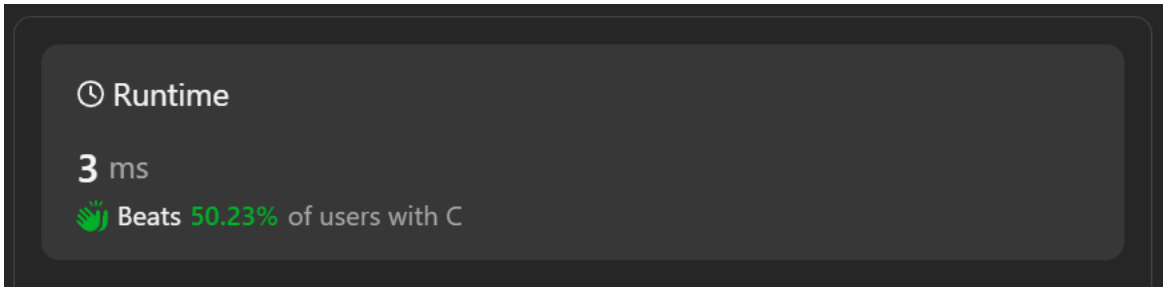
```
void append(struct ListNode** head,int val){
    struct ListNode *new_node=(struct ListNode*)malloc(sizeof(struct ListNode));
    new_node->val=val;
    new_node->next=NULL;
    struct ListNode *prev=*head;
    if(prev==NULL){
        *head=new_node;
        return;
    }
```

```
    while(prev->next!=NULL){
        prev=prev->next;
    }
    prev->next=new_node;
}

void mid(struct ListNode *prev,int val){
    struct ListNode *new_node=(struct ListNode *)malloc(sizeof(struct ListNode));
    new_node->val=val;
    new_node->next=prev->next;
    prev->next=new_node;
}

void push(struct ListNode **head,int value){
    struct ListNode *new_node=(struct ListNode *)malloc(sizeof(struct ListNode));
    new_node->val=value;
    new_node->next=(*head);
    *head=new_node;
}

struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    struct ListNode *newhead=NULL;
    int i=1;
    struct ListNode *cur=head;
    struct ListNode *prev=newhead;
    while(cur!=NULL){
        if(left<=i && right>=i){
            if(prev==NULL){
                append(&newhead,cur->val);
                prev=newhead;
```

```
            cur=cur->next;
            i++;
            continue;
        }
        else if(left==1){
            push(&newhead,cur->val);
        }
        else{
            mid(prev,cur->val);
        }
    }
    else{
        append(&newhead,cur->val);
        prev=(prev==NULL)?newhead:prev->next;
    }
    i++;
    cur=cur->next;
  }
  return newhead;
}

void display(struct ListNode *head){
  if(head==NULL){
    printf("Linked List is empty.\n");
    return;
  }
  printf("Linked List:");
  while(head!=NULL){
    printf("%d ",head->val);
    head=head->next;
```

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

Output:



Week 5 (25-01-2024):

Q1)

Write a C program to demonstrate sort, reverse and concatenation of singly linked list.

Ans)

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

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

void sort_list(struct Node *head)
{
    struct Node *p,*q;
    int temp;
    for(p=head;p!=NULL;p=p->next)
    {
        for(q=p->next;q!=NULL;q=q->next)
```

```
        {
            if(p->data>q->data)
            {
                temp=q->data;
                q->data=p->data;
                p->data=temp;
            }
        }
    }
}

void reverse_list(struct Node **head)
{
    struct Node *cur=*head;
    struct Node *pre=NULL;
    struct Node *next=NULL;
    while(cur!=NULL)
    {
        next=cur->next;
        cur->next=pre;
        pre=cur;
        cur=next;
    }
    *head=pre;
}

void concat_list(struct Node *head1,struct Node *head2)
{
    struct Node *head3;
    struct Node *temp;
```

```
    if(head1==NULL)
    {
        head3=head2;
        display_list(head3);
    }
    else if(head2==NULL)
    {
        head3=head1;
        display_list(head3);
    }
    else
    {
        temp=head1;
        head3=head1;
        while(temp->next!=NULL)
            temp=temp->next;
        temp->next=head2;
        display_list(head3);
    }
}

void display_list(struct Node* head)
{
    struct Node* current=head;
    if(current==NULL)
    {
        printf("The list is empty\n");
        return;
    }
    else
```

```
    {
        printf("The contents of the list are:\n");
        while(current!=NULL)
        {
            printf("%d",current->data);
            current=current->next;
        }
        printf("\n");
    }
}

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

void insertEnd(struct Node** head, int data)
{
    struct Node* newNode=createLinkedList(data);

    if(*head==NULL)
    {
        *head=newNode;
    }
    else
    {
        struct Node* current=*head;
```

```
        while(current->next != NULL)
        {
            current=current->next;
        }
        current->next=newNode;
    }
}

int main()
{
    struct Node* head = NULL;
    insertEnd(&head,3);
    insertEnd(&head,1);
    insertEnd(&head,4);
    insertEnd(&head,2);
    insertEnd(&head,6);
    insertEnd(&head,5);
    struct Node* head1 = NULL;
    insertEnd(&head1,1);
    insertEnd(&head1,3);
    insertEnd(&head1,4);
    insertEnd(&head1,2);
    struct Node* head2 = NULL;
    insertEnd(&head2,3);
    insertEnd(&head2,1);
    insertEnd(&head2,4);
    insertEnd(&head2,2);
    struct Node* head3 = NULL;
    printf("Initial List:\n");
    display_list(head);
```

```
    int c,c1=0;
    while(c1==0)
    {
        printf("\nEnter 1 to sort the linked list\n");
        printf("Enter 2 to reverse the linked list\n");
        printf("Enter 3 to concatenate 2 linked lists\n");
        printf("Enter 4 to display\n");
        printf("Enter 0 to exit\n");
        printf("\nEnter your choice:\n");
        scanf("%d",&c);
        switch(c)
        {
            case 0: printf("Exiting...\n");c1=1;break;
            case 1: sort_list(head);display_list(head);break;
            case 2: reverse_list(&head);display_list(head);break;
            case 3:
                printf("The 2 lists being concatenated are:\n");
                display_list(head1);
                printf("and\n");
                display_list(head2);
                printf("Concatenated list:\n");
                concat_list(head1,head2);break;
            case 4: display_list(head);break;
            default: printf("Invalid choice. Please enter a valid option\n");break;
        }
    }
    return 0;
}
```

Output:

```
Initial List:
The contents of the list are:
314265

Enter 1 to sort the linked list
Enter 2 to reverse the linked list
Enter 3 to concatenate 2 linked lists
Enter 4 to display
Enter 0 to exit

Enter your choice:
1
The contents of the list are:
123456

Enter 1 to sort the linked list
Enter 2 to reverse the linked list
Enter 3 to concatenate 2 linked lists
Enter 4 to display
Enter 0 to exit

Enter your choice:
2
The contents of the list are:
654321
```

```
Enter 1 to sort the linked list
Enter 2 to reverse the linked list
Enter 3 to concatenate 2 linked lists
Enter 4 to display
Enter 0 to exit

Enter your choice:
3
The 2 lists being concatenated are:
The contents of the list are:
1342
and
The contents of the list are:
3142
Concatenated list:
The contents of the list are:
13423142

Enter 1 to sort the linked list
Enter 2 to reverse the linked list
Enter 3 to concatenate 2 linked lists
Enter 4 to display
Enter 0 to exit

Enter your choice:
4
The contents of the list are:
654321
```

Q2)

Write a C program to demonstrate Stack and Queue operations using Singly linked lists.

Ans)

```
#include<stdio.h>

#include<stdlib.h>

struct Node

{

   int data;

   struct Node *next;

};


void push_list(struct Node **head,int new_data)

{
```

```
    struct Node *new_node=(struct Node *)malloc(sizeof(struct Node));
    new_node->data=new_data;
    new_node->next=NULL;
    struct Node *last=*head;
    if(*head==NULL)
        *head=new_node;
    else
    {
        while(last->next!=NULL)
            last=last->next;
        last->next=new_node;
    }
}

void pop_listStack(struct Node *head)
{
    if(head==NULL)
    {
        printf("List is empty\n");
        return;
    }
    struct Node *last=head;
    struct Node *pre;
    while(last->next!=NULL)
    {
        pre=last;
        last=last->next;
    }
    free(last);
    pre->next=NULL;
```

```
}

void enqueue_list(struct Node **head,int new_data)
{
   struct Node *new_node=(struct Node *)malloc(sizeof(struct Node));
   new_node->data=new_data;
   new_node->next=NULL;
   struct Node *last=*head;
   if(*head==NULL)
      *head=new_node;
   else
   {
      while(last->next!=NULL)
         last=last->next;
      last->next=new_node;
   }
}

void dequeue_list(struct Node **head)
{
   if(*head==NULL)
   {
      printf("List is empty\n");
      return;
   }
   struct Node *temp=(*head)->next;
   free(*head);
   *head=temp;
}
```

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

void display_listQueue(struct Node *head)
{
   if(head==NULL)
   {
      printf("List is empty\n");
      return;
   }
   printf("Queue:\n");
   while(head!=NULL)
   {
      printf("%d",head->data);
      head=head->next;
   }
   printf("\n");
```

```
}

int main()
{
   struct Node *head=NULL;
   struct Node *head1=NULL;
   int c,c1=0,elem1,elem2;
   while(c1==0)
   {
      printf("\nEnter 1 to push\n");
      printf("Enter 2 to pop\n");
      printf("Enter 3 to display stack list\n");
      printf("Enter 4 to enqueue\n");
      printf("Enter 5 to dequeue\n");
      printf("Enter 6 to display queue list\n");
      printf("Enter 0 to exit\n");
      printf("\nEnter your choice:\n");
      scanf("%d",&c);
      switch(c)
      {
         case 0: printf("Exiting...\n");c1=1;break;
         case 1:
            printf("Enter the value to push:\n");
            scanf("%d",&elem1);
            push_list(&head,elem1);
            display_listStack(head);
            break;
         case 2:
            pop_listStack(head);
            display_listStack(head);
```

```
            break;
        case 3: display_listStack(head);break;
        case 4:
            printf("Enter the value to enqueue:\n");
            scanf("%d",&elem2);
            enqueue_list(&head1,elem2);
            display_listQueue(head1);
            break;
        case 5:
            dequeue_list(&head1);
            display_listQueue(head1);
            break;
        case 6: display_listQueue(head1);break;
        default: printf("Invalid choice.\n");break;
      }
   }
   return 0;
}
```

Output:

```
Enter 1 to push
Enter 2 to pop
Enter 3 to display stack list
Enter 4 to enqueue
Enter 5 to dequeue
Enter 6 to display queue list
Enter 0 to exit

Enter your choice:
1
Enter the value to push:
1
Stack:
1

Enter 1 to push
Enter 2 to pop
Enter 3 to display stack list
Enter 4 to enqueue
Enter 5 to dequeue
Enter 6 to display queue list
Enter 0 to exit

Enter your choice:
1
Enter the value to push:
2
Stack:
12
```

```
Enter 1 to push
Enter 2 to pop
Enter 3 to display stack list
Enter 4 to enqueue
Enter 5 to dequeue
Enter 6 to display queue list
Enter 0 to exit

Enter your choice:
4
Enter the value to enqueue:
1
Queue:
1

Enter 1 to push
Enter 2 to pop
Enter 3 to display stack list
Enter 4 to enqueue
Enter 5 to dequeue
Enter 6 to display queue list
Enter 0 to exit

Enter your choice:
4
Enter the value to enqueue:
2
Queue:
12
```

```
Enter 1 to push
Enter 2 to pop
Enter 3 to display stack list
Enter 4 to enqueue
Enter 5 to dequeue
Enter 6 to display queue list
Enter 0 to exit

Enter your choice:
2
Stack:
1

Enter 1 to push
Enter 2 to pop
Enter 3 to display stack list
Enter 4 to enqueue
Enter 5 to dequeue
Enter 6 to display queue list
Enter 0 to exit

Enter your choice:
3
Stack:
1
```

```
Enter 1 to push
Enter 2 to pop
Enter 3 to display stack list
Enter 4 to enqueue
Enter 5 to dequeue
Enter 6 to display queue list
Enter 0 to exit

Enter your choice:
5
Queue:
2

Enter 1 to push
Enter 2 to pop
Enter 3 to display stack list
Enter 4 to enqueue
Enter 5 to dequeue
Enter 6 to display queue list
Enter 0 to exit

Enter your choice:
6
Queue:
2

Enter 1 to push
Enter 2 to pop
Enter 3 to display stack list
Enter 4 to enqueue
Enter 5 to dequeue
Enter 6 to display queue list
Enter 0 to exit

Enter your choice:
Queue:
2
```

Week 6 (01-02-2024):

Q1)

Write a C program to demonstrate Doubly Linked List – Creation, insertion to the left of a node, deletion of a node and display implementations.

Ans)

```
#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)
    {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data=value;
    newNode->prev=NULL;
    newNode->next=NULL;
    return newNode;
}

void insertLeft(struct Node** head,int value,int targetValue)
{
    struct Node* newNode=createNode(value);
```

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

void deleteNode(struct Node** head,int value)
{
    struct Node* current=*head;
    while(current!=NULL && current->data!=value)
    {
        current=current->next;
```

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

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

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

```
}

int main()
{
   struct Node* head=NULL;
   int choice,value,targetValue;
   do
   {
      printf("\nEnter 1 to create a doubly linked list\n");
      printf("Enter 2 to insert a node to the left of a node\n");
      printf("Enter 3 to delete a node based on a specific value\n");
      printf("Enter 4 to display the contents of the list\n");
      printf("Enter 5 to exit\n");
      printf("\nEnter your choice: \n");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
            if (head!=NULL)
            {
               printf("Doubly linked list already created.\n");
            }
            else
            {
               printf("Enter the value for the node: ");
               scanf("%d",&value);
               head=createNode(value);
               printf("Doubly linked list created successfully.\n");
            }
            break;
```

```
case 2:
   if(head==NULL)
   {
      printf("List is empty. Please create a list first.\n");
   }
   else
   {
      printf("Enter the value to insert:\n");
      scanf("%d",&value);
      printf("Enter the value of the node to the left of which to insert:\n");
      scanf("%d",&targetValue);
      insertLeft(&head,value,targetValue);
   }
   break;
case 3:
   if(head==NULL)
   {
      printf("List is empty. Please create a list first.\n");
   }
   else
   {
      printf("Enter the value to delete:\n");
      scanf("%d",&value);
      deleteNode(&head,value);
   }
   break;
case 4:
   if(head==NULL)
   {
      printf("List is empty. Please create a list first.\n");
```

```
            }
            else
            {
                printf("The contents of the list are:\n");
                displayList(head);
            }
            break;
        case 5:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please enter a valid option.\n");break;
      }
   } while(choice!=5);
   return 0;
}
```

Output:

```
Enter 1 to create a doubly linked list
Enter 2 to insert a node to the left of a node
Enter 3 to delete a node based on a specific value
Enter 4 to display the contents of the list
Enter 5 to exit

Enter your choice:
1
Enter the value for the node: 1
Doubly linked list created successfully.

Enter 1 to create a doubly linked list
Enter 2 to insert a node to the left of a node
Enter 3 to delete a node based on a specific value
Enter 4 to display the contents of the list
Enter 5 to exit

Enter your choice:
2
Enter the value to insert:
2
Enter the value of the node to the left of which to insert:
1

Enter 1 to create a doubly linked list
Enter 2 to insert a node to the left of a node
Enter 3 to delete a node based on a specific value
Enter 4 to display the contents of the list
Enter 5 to exit

Enter your choice:
4
The contents of the list are:
21
```

```
Enter 1 to create a doubly linked list
Enter 2 to insert a node to the left of a node
Enter 3 to delete a node based on a specific value
Enter 4 to display the contents of the list
Enter 5 to exit

Enter your choice:
3
Enter the value to delete:
1

Enter 1 to create a doubly linked list
Enter 2 to insert a node to the left of a node
Enter 3 to delete a node based on a specific value
Enter 4 to display the contents of the list
Enter 5 to exit

Enter your choice:
4
The contents of the list are:
2
```

Q2)

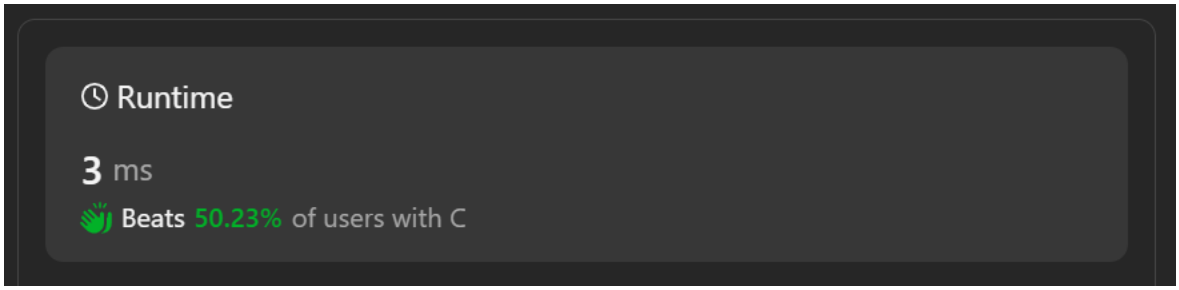
Leet Code – Singly Linked List

Ans)

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
    struct ListNode **ans = (struct ListNode **)calloc(1, sizeof(struct ListNode *) * k);
    struct ListNode *prev;
    int base, len = 0, part = 0;
    for (struct ListNode *tmp = head;tmp!=NULL;tmp = tmp->next) {
        len++;
    }
    base = len / k;
    for (int i = len % k; i > 0; i--) {
        ans[part] = head;
        part++;
        for (int i = 0; i < (base + 1); i++) {
            prev = head;
            head = head->next;
        }
        prev->next = NULL;
    }
```

```
    if (base!=0) {
        for (int i = part; i < k; i++) {
            ans[part] = head;
            part++;
            for (int j = 0; j < base; j++) {
                prev = head;
                head = head->next;
            }
            prev->next = NULL;
        }
    }
    *returnSize = k;
    return ans;
}
```

Output:



Week 7 (15-02-2024):

Q1)

Write a C program
a. To construct a binary Search tree.
b. To traverse the tree using all the methods i.e., in-order, preorder and postorder
c. To display the elements in the tree.

Ans)

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

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

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

struct Node* insert(struct Node* root,int value)
{
   if(root==NULL)
      return createNode(value);
   if(value<root->data)
      root->left=insert(root->left,value);
   else if(value>root->data)
      root->right=insert(root->right,value);
   return root;
}

void inOrderTraversal(struct Node* root)
{
   if(root!=NULL)
   {
```

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

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

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

void display(struct Node* root)
{
    printf("Elements in the tree: ");
    inOrderTraversal(root);
```

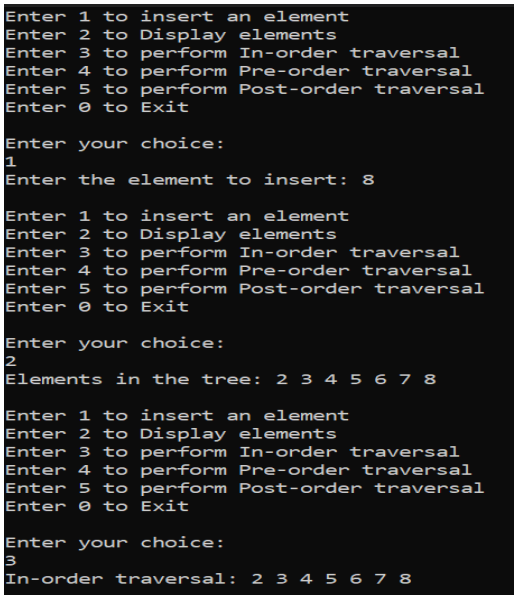
```
    printf("\n");
}

int main()
{
    struct Node* root=NULL;
    int c,value;
    do {
        printf("\nEnter 1 to insert an element\n");
        printf("Enter 2 to Display elements\n");
        printf("Enter 3 to perform In-order traversal\n");
        printf("Enter 4 to perform Pre-order traversal\n");
        printf("Enter 5 to perform Post-order traversal\n");
        printf("Enter 0 to Exit\n");
        printf("\nEnter your choice:\n");
        scanf("%d",&c);
        switch(c)
        {
            case 1:
                printf("Enter the element to insert: ");
                scanf("%d",&value);
                root=insert(root,value);
                break;
            case 2:
                display(root);
                break;
            case 3:
                printf("In-order traversal: ");
                inOrderTraversal(root);
                printf("\n");
```

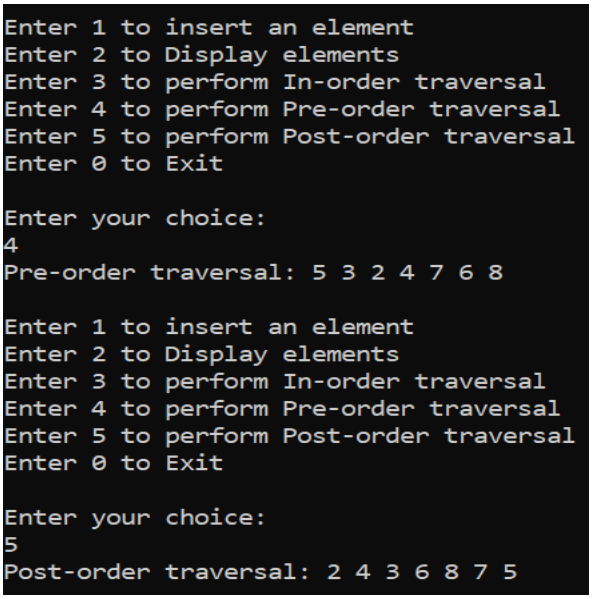
```
            break;
        case 4:
            printf("Pre-order traversal: ");
            preOrderTraversal(root);
            printf("\n");
            break;
        case 5:
            printf("Post-order traversal: ");
            postOrderTraversal(root);
            printf("\n");
            break;
        case 0:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
  }while(c!=0);
  return 0;
}
```

Output:

```
Enter 1 to insert an element
Enter 2 to Display elements
Enter 3 to perform In-order traversal
Enter 4 to perform Pre-order traversal
Enter 5 to perform Post-order traversal
Enter 0 to Exit

Enter your choice:
1
Enter the element to insert: 5

Enter 1 to insert an element
Enter 2 to Display elements
Enter 3 to perform In-order traversal
Enter 4 to perform Pre-order traversal
Enter 5 to perform Post-order traversal
Enter 0 to Exit

Enter your choice:
1
Enter the element to insert: 3

Enter 1 to insert an element
Enter 2 to Display elements
Enter 3 to perform In-order traversal
Enter 4 to perform Pre-order traversal
Enter 5 to perform Post-order traversal
Enter 0 to Exit

Enter your choice:
1
Enter the element to insert: 7
```

```
Enter 1 to insert an element
Enter 2 to Display elements
Enter 3 to perform In-order traversal
Enter 4 to perform Pre-order traversal
Enter 5 to perform Post-order traversal
Enter 0 to Exit

Enter your choice:
1
Enter the element to insert: 2

Enter 1 to insert an element
Enter 2 to Display elements
Enter 3 to perform In-order traversal
Enter 4 to perform Pre-order traversal
Enter 5 to perform Post-order traversal
Enter 0 to Exit

Enter your choice:
1
Enter the element to insert: 4

Enter 1 to insert an element
Enter 2 to Display elements
Enter 3 to perform In-order traversal
Enter 4 to perform Pre-order traversal
Enter 5 to perform Post-order traversal
Enter 0 to Exit

Enter your choice:
1
Enter the element to insert: 6
```





Q2)

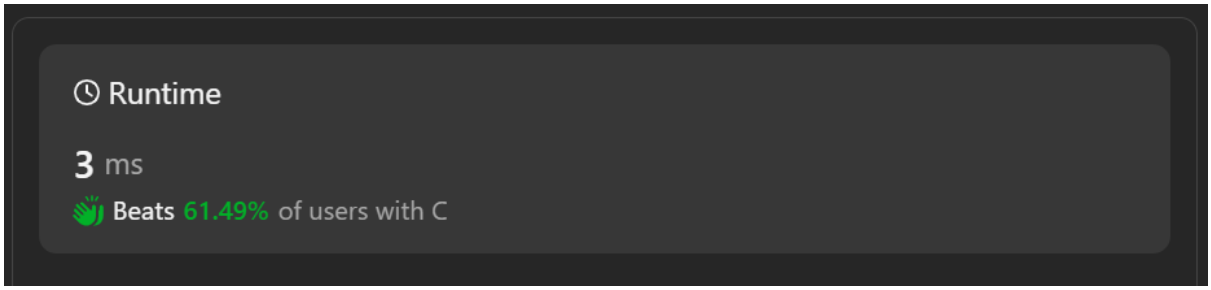
Leet Code – Singly Linked List

Ans)

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* rotateRight(struct ListNode* head, int k) {
    if(head==NULL||k==0){
        return head;
    }
    int n=0;
    struct ListNode* last=head;
    while(last->next!=NULL){
        n++;
        last=last->next;
    }
```

```
    n++;
    last->next=head;
    int rotate=(n-(k%n));
    for(int i=0;i<rotate;i++){
        head=head->next;
    }
    struct ListNode* ptr1=head;
    while(ptr1->next!=head){
        ptr1=ptr1->next;
    }
    ptr1->next=NULL;
    return head;
}
```

Output:



Week 8 (22-02-2024):

Q1)

Write a C program to traverse a graph using BFS method and to check whether a graph is connected or not using DFS method.

Ans)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
struct Queue
{
```

```
    int front,rear,size;
    unsigned capacity;
    int* array;
};

struct Queue* createQueue(unsigned capacity)
{
    struct Queue* queue=(struct Queue*)malloc(sizeof(struct Queue));
    queue->capacity=capacity;
    queue->front=queue->size=0;
    queue->rear=capacity-1;
    queue->array=(int*)malloc(queue->capacity*sizeof(int));
    return queue;
}

int isEmpty(struct Queue* queue)
{
    return (queue->size==0);
}

void enqueue(struct Queue* queue,int item)
{
    if (isFull(queue))
        return;
    queue->rear=(queue->rear+1)%queue->capacity;
    queue->array[queue->rear]=item;
    queue->size=queue->size+1;
}

int dequeue(struct Queue* queue)
```

```
{
    if (isEmpty(queue))
        return -1;
    int item=queue->array[queue->front];
    queue->front=(queue->front+1)%queue->capacity;
    queue->size=queue->size-1;
    return item;
}

int isFull(struct Queue* queue)
{
    return (queue->size==queue->capacity);
}

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

struct Graph* createGraph(int numVertices)
{
    struct Graph* graph=(struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices=numVertices;
    graph->adjMatrix=(int**)malloc(numVertices*sizeof(int*));
    for(int i=0;i<numVertices;i++)
    {
        graph->adjMatrix[i]=(int*)malloc(numVertices*sizeof(int));
    }
    for(int i=0;i<numVertices;i++)
```

```
    {
        for(int j=0;j<numVertices;j++)
        {
            graph->adjMatrix[i][j]=0;
        }
    }
    return graph;
}

void addEdge(struct Graph* graph,int src,int dest)
{
    graph->adjMatrix[src][dest]=1;
    graph->adjMatrix[dest][src]=1;
}

void BFS(struct Graph* graph,int startVertex)
{
    struct Queue* queue=createQueue(MAX_VERTICES);
    int visited[MAX_VERTICES];
    for (int i=0;i<MAX_VERTICES;i++)
    {
        visited[i]=0;
    }
    visited[startVertex]=1;
    enqueue(queue,startVertex);
    while(!isEmpty(queue))
    {
        int currentVertex=dequeue(queue);
        printf("%d ",currentVertex);
        for(int i=0;i<graph->numVertices;i++)
```

```
        {
            if(graph->adjMatrix[currentVertex][i]==1 && !visited[i])
            {
                visited[i]=1;
                enqueue(queue,i);
            }
        }
    }
    free(queue);
}

void DFSUtil(struct Graph* graph,int vertex,int visited[])
{
    visited[vertex]=1;
    printf("%d ",vertex);
    for(int i=0;i<graph->numVertices;i++)
    {
        if(graph->adjMatrix[vertex][i]==1 && !visited[i])
        {
            DFSUtil(graph,i,visited);
        }
    }
}

int isConnected(struct Graph* graph)
{
    int visited[MAX_VERTICES];
    for (int i=0;i<MAX_VERTICES;i++)
    {
        visited[i]=0;
```

```
    }
    DFSUtil(graph,0,visited);
    for(int i=0;i<graph->numVertices;i++)
    {
        if(!visited[i])
        {
            return 0;
        }
    }
    return 1;
}

int main()
{
    struct Graph* graph=createGraph(5);
    addEdge(graph,0,1);
    addEdge(graph,0,2);
    addEdge(graph,1,3);
    addEdge(graph,2,4);
    printf("BFS traversal: ");
    BFS(graph,0);
    printf("\n");
    if(isConnected(graph))
    {
        printf("The graph is connected.\n");
    }
    else
    {
        printf("The graph is not connected.\n");
    }
```

```
    for(int i=0; i<graph->numVertices;i++)
    {
        free(graph->adjMatrix[i]);
    }
    free(graph->adjMatrix);
    free(graph);
    return 0;
}
```

Output:

```
BFS traversal: 0 1 2 3 4
0 1 3 2 4 The graph is connected.
```

Q2)

Hacker Rank on tree.

Ans)

```
#include<stdio.h>
#include<stdlib.h>
struct node {
    int data;
    struct node *left;
    struct node *right;
};
struct node* create_node(int val){
    if(val == -1){
        return NULL;
    }
    struct node *temp=(struct node*)malloc(sizeof(struct node));
    temp->data=val;
    temp->left=NULL;
    temp->right=NULL;
    return temp;
```

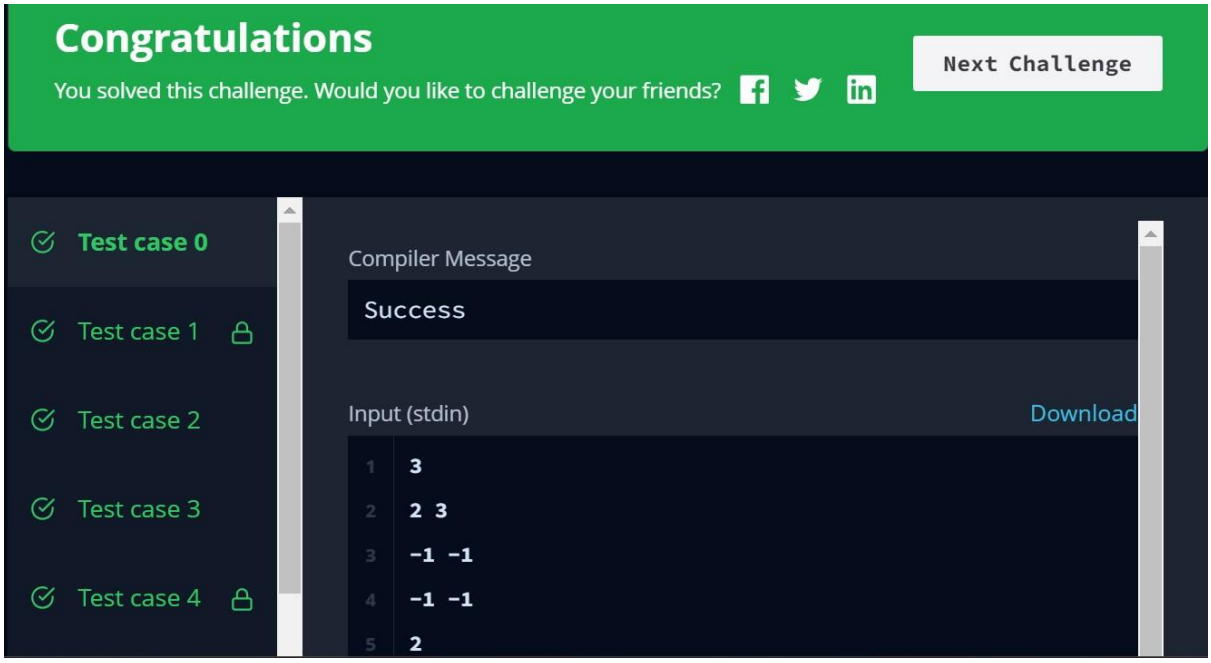
```
}
void inorder(struct node *root){
   if(!root){
      return;
   }
   inorder(root->left);
   printf("%d ", root->data);
   inorder(root->right);
}
int max(int a, int b){
   if(a>b){
      return a;
   } else {
      return b;
   }
}
int height(struct node * root){
   if(!root){
      return 0;
   }
   return(1+max(height(root->left),height(root->right)));
}

void swap_nodes_at_level(struct node *root, int inc, int level, int height){
   struct node *tnode;
   if(!root){
      return;
   }
   if(level > height){
      return;
```

```
    }
    if(!(level%inc)){
        tnode=root->left;
        root->left=root->right;
        root->right=tnode;
    }
    swap_nodes_at_level(root->left, inc, level+1, height);
    swap_nodes_at_level(root->right, inc, level+1, height);
}
int tail=0;
int head=0;
void enqueue(struct node **queue, struct node *root){
    queue[tail]=root;
    tail++;
}
struct node* dequeue(struct node **queue){
    struct node *temp = queue[head];
    head++;
    return temp;
}
int main() {
    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    int nodes_count, i, temp, h, tc_num, index, inc, temp1, temp2;
    scanf("%d", &nodes_count);
  //  printf("%d\n", nodes_count);
    // int arr[2*nodes_count+1];
    struct node *root_perm, *root_temp;
    //queue=create_queue(nodes_count);
    struct node *q[nodes_count];
    for(i=0;i<nodes_count;i++){
```

```
    q[i]=NULL;
}
//building the array
i=0,index=1;
root_temp=root_perm=create_node(1);
enqueue(q, root_temp);
while(index<=2*nodes_count) {
    //printf("\n In Loop : i : %d",i);
    root_temp=dequeue(q);
    //setting up the left child
    scanf("%d", &temp1);
    if(temp1 == -1){
    } else {
        root_temp->left=create_node(temp1);
        enqueue(q, root_temp->left);
    }
    //setting up the right child
    scanf("%d", &temp2);
    if(temp2==-1) {
    } else {
        root_temp->right=create_node(temp2);
        enqueue(q, root_temp->right);
    }
    index=index+2;
  //  i++;
}
h = height(root_perm);
scanf("%d", &tc_num);
//printf("%d",tc_num);
//printf("\n");
```

```
    //inorder(root_perm);
    while(tc_num){
        scanf("%d",&inc);
        temp=inc;
        //while(temp < height){
        swap_nodes_at_level(root_perm, inc, 1, h);
        //temp=temp + inc;
        //}
        //temp=0;
        inorder(root_perm);
        printf("\n");
        tc_num--;
    }
    //Tree is created at this point
    return 0;
}
```

Output:



Week 9 (29-02-2024):

Q1)

Write a C program that uses Hash function H: K->L as H(K)=K mod m (remainder method) and implement hashing technique to map a given key K to the address space L. Resolve collision if any.

Ans)

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

#define TABLE_SIZE 10

// Function to calculate hash using remainder method
int hashFunction(int key, int m) {
    return key % m;
}

// Function to insert a value using linear probing
void insert(int hashtable[], int key, int m) {
    int i = 0;
    int hkey = hashFunction(key, m);
    int index;

    do {
        index = (hkey + i) % m;
        if (hashtable[index] == -1) {
            // If the slot is empty, insert the key
            hashtable[index] = key;
            printf("Inserted key %d at index %d\n", key, index);
            return;
        }
        i++;
    } while (i < m);
```

```
    printf("Unable to insert key %d. Table is full.\n", key);
}

// Function to search a value using linear probing
void search(int hashtable[], int key, int m) {
    int i = 0;
    int hkey = hashFunction(key, m);
    int index;

    do
    {
        index = (hkey + i) % m;
        if (hashtable[index] == key)
        {
            // If the key is found at the calculated index
            printf("Key %d found at index %d\n", key, index);
            return;
        }
        i++;
    }while(i<m);

    printf("Key %d not found in the table.\n", key);
}

int main()
{
    int hashtable[TABLE_SIZE];
    int i;

    // Initialize hashtable with -1 indicating empty slots
```

```
    for (i = 0; i < TABLE_SIZE; i++)
    {
        hashtable[i] = -1;
    }

    // Inserting values into the hashtable
    insert(hashtable, 1234, TABLE_SIZE);
    insert(hashtable, 5678, TABLE_SIZE);
    insert(hashtable, 9012, TABLE_SIZE);
    insert(hashtable, 2318, TABLE_SIZE);

    // Searching for values in the hashtable
    search(hashtable, 5678, TABLE_SIZE);
    search(hashtable, 2318, TABLE_SIZE);
    search(hashtable, 9999, TABLE_SIZE); // Not present in the hashtable

    return 0;
}
```

Output:

```
Inserted key 1234 at index 4
Inserted key 5678 at index 8
Inserted key 9012 at index 2
Inserted key 2318 at index 9
Key 5678 found at index 8
Key 2318 found at index 9
Key 9999 not found in the table.
```